CS433_Assignment 2:

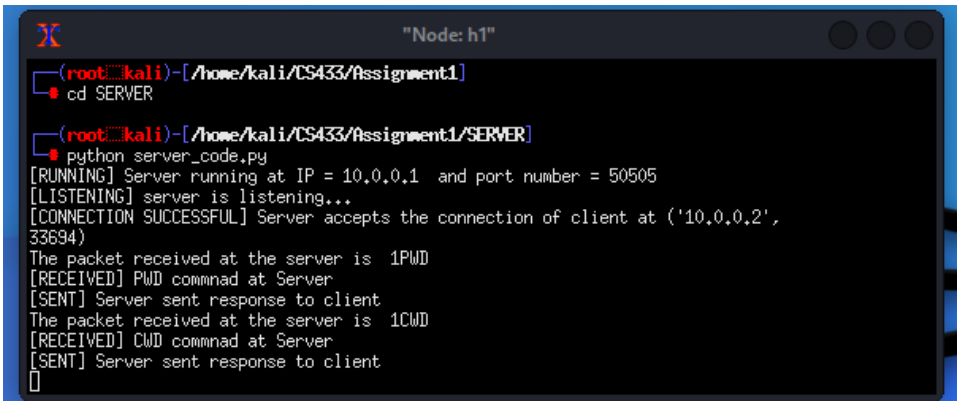
Name: Chirag Sarda

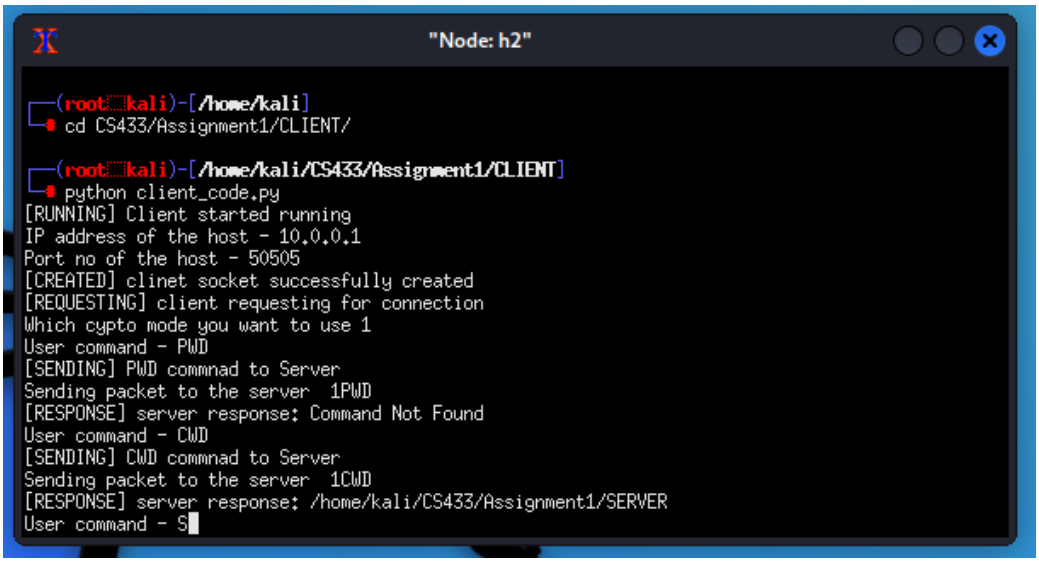
Roll No.: 20110047

Question 1

A

I just hardcoded the IP address of the server in the code (here in my case it is h1). Then just put that IP address on the client side. It gets connected automatically. For getting the IP address of the server I first did the $ifconfig$ at the server





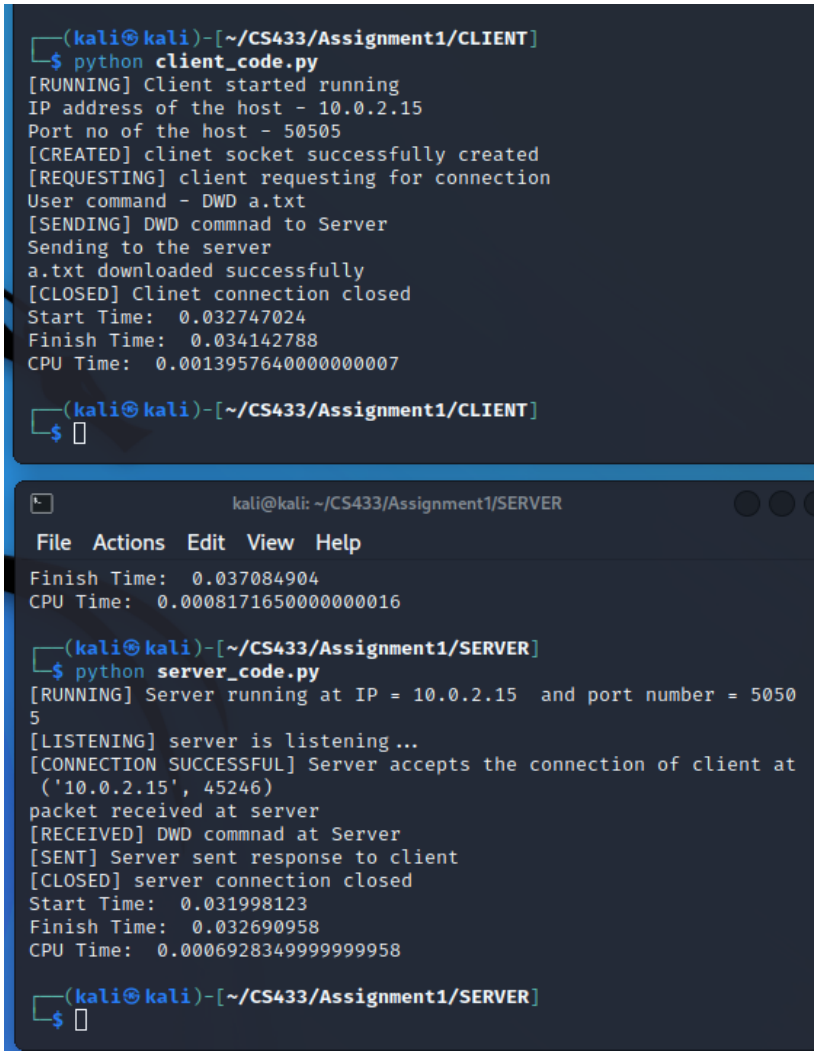
(B)

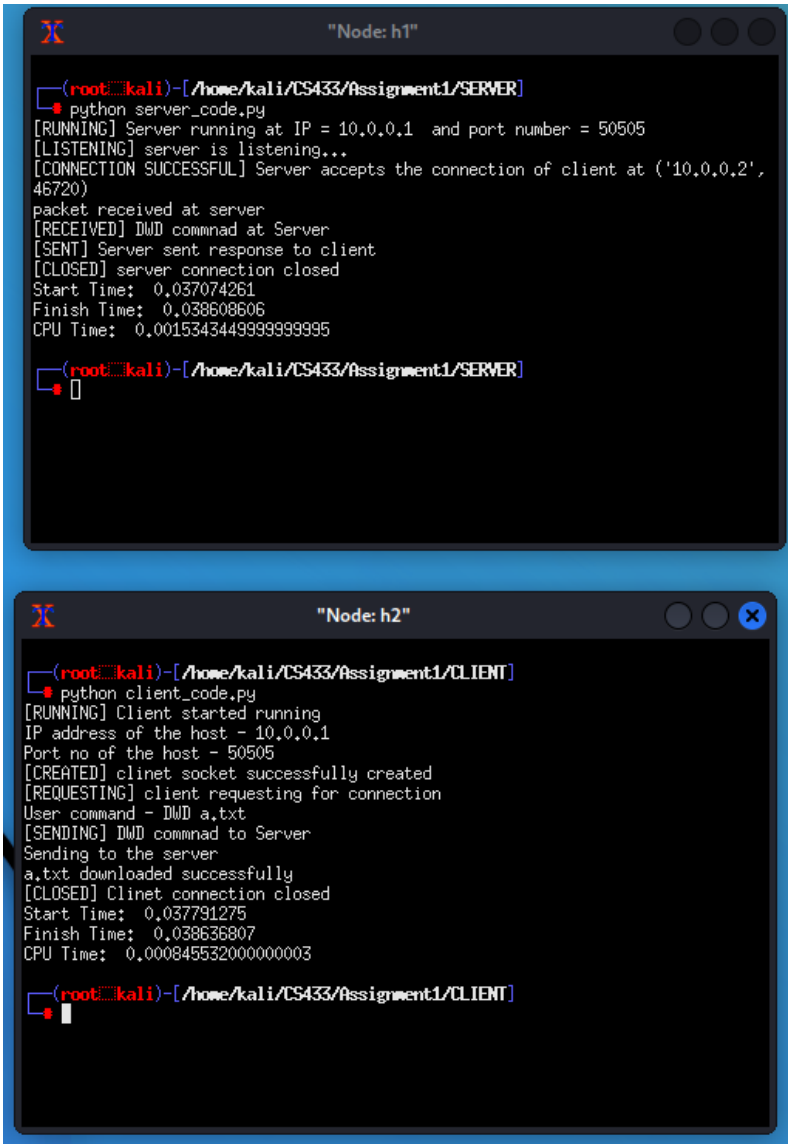
```
┌──(kali㉿kali)-[~/CS433/Assignment1/CLIENT]
└─$ python client_code.py
[RUNNING] Client started running
IP address of the host - 10.0.2.15
Port no of the host - 50505
[CREATED] clinet socket successfully created
[REQUESTING] client requesting for connection
User command - DWD a.txt
[SENDING] DWD commnad to Server
Sending to the server
a.txt downloaded successfully
[CLOSED] Clinet connection closed
Start Time:  0.035755359
Finish Time:  0.038266963
CPU Time:  0.0025116040000000006

┌──(kali㉿kali)-[~/CS433/Assignment1/CLIENT]
└─$
```

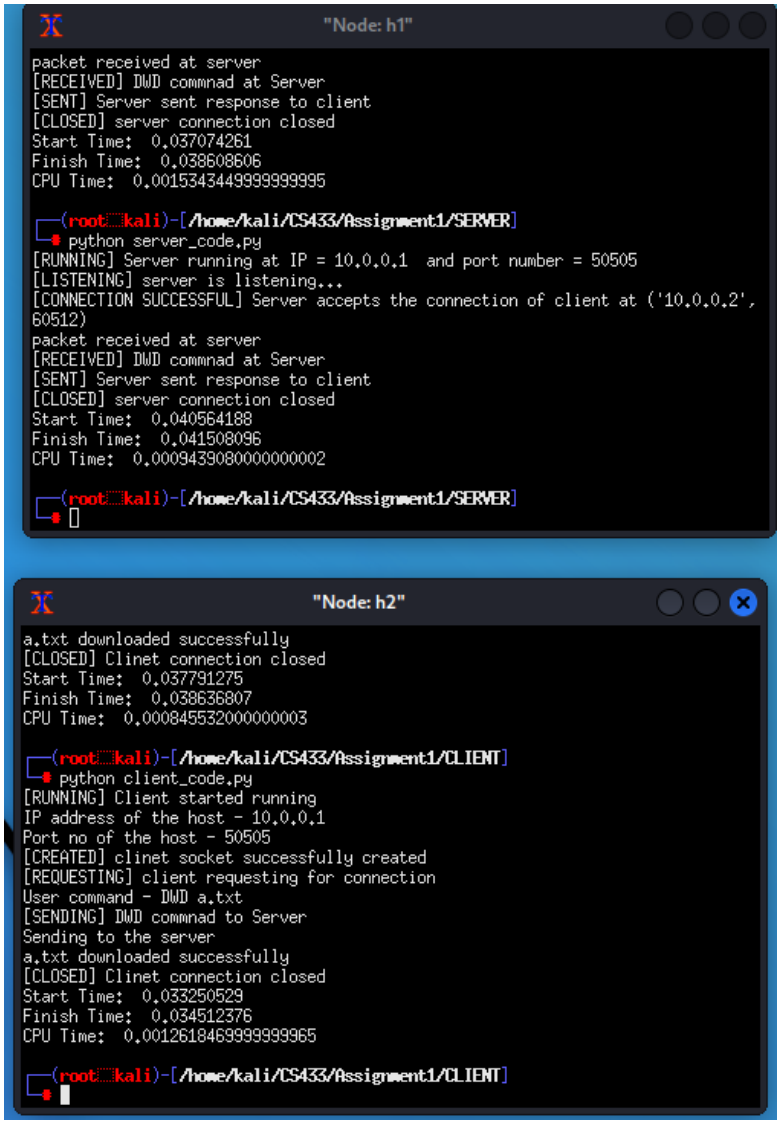
```
kali@kali: ~/CS433/Assignment1/SERVER
File  Actions  Edit  View  Help
sys     0.00s
cpu     0%

┌──(kali㉿kali)-[~/CS433/Assignment1/SERVER]
└─$ python server_code.py
[RUNNING] Server running at IP = 10.0.2.15  and port number = 5050
5
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at
 ('10.0.2.15', 48616)
packet received at server
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client
[CLOSED] server connection closed
Start Time:  0.034208495
Finish Time:  0.035272161
CPU Time:  0.0010636660000000048

┌──(kali㉿kali)-[~/CS433/Assignment1/SERVER]
└─$
```

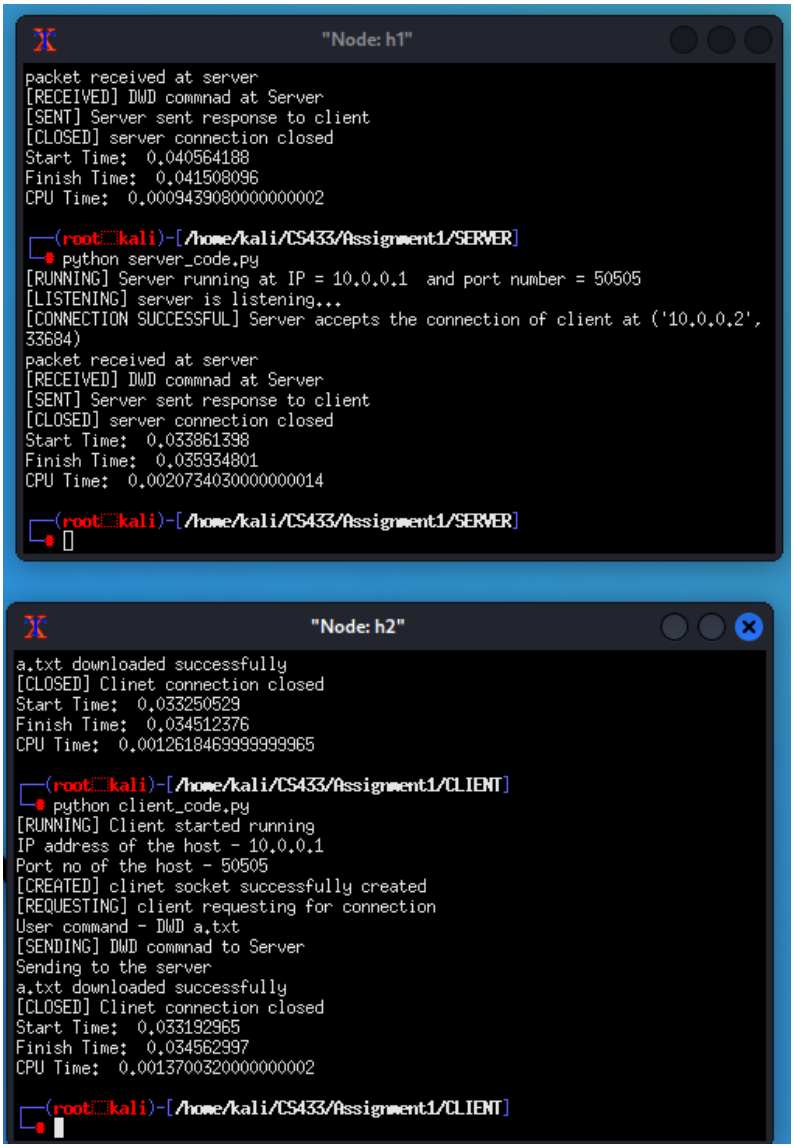
```
┌──(kali㉿kali)-[~/CS433/Assignment1/CLIENT]
└─$ python client_code.py
[RUNNING] Client started running
IP address of the host - 10.0.2.15
Port no of the host - 50505
[CREATED] clinet socket successfully created
[REQUESTING] client requesting for connection
User command - DWD a.txt
[SENDING] DWD commnad to Server
Sending to the server
a.txt downloaded successfully
[CLOSED] Clinet connection closed
Start Time:  0.034533659
Finish Time:  0.036137693
CPU Time:  0.0016040339999999972

┌──(kali㉿kali)-[~/CS433/Assignment1/CLIENT]
└─$
```

```
kali@kali: ~/CS433/Assignment1/SERVER
File  Actions  Edit  View  Help
Finish Time:  0.035272161
CPU Time:  0.0010636660000000048

┌──(kali㉿kali)-[~/CS433/Assignment1/SERVER]
└─$ python server_code.py
[RUNNING] Server running at IP = 10.0.2.15  and port number = 5050
5
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at
 ('10.0.2.15', 47860)
packet received at server
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client
[CLOSED] server connection closed
Start Time:  0.036267739
Finish Time:  0.037084904
CPU Time:  0.0008171650000000016

┌──(kali㉿kali)-[~/CS433/Assignment1/SERVER]
└─$
```


(kali㉿kali)-[~/CS433/Assignment1/CLIENT]
$ python client_code.py
[RUNNING] Client started running
IP address of the host - 10.0.2.15
Port no of the host - 50505
[CREATED] clinet socket successfully created
[REQUESTING] client requesting for connection
User command - DWD a.txt
[SENDING] DWD commnad to Server
Sending to the server
a.txt downloaded successfully
[CLOSED] Clinet connection closed
Start Time: 0.032747024
Finish Time: 0.034142788
CPU Time: 0.0013957640000000007
(kali㉿kali)-[~/CS433/Assignment1/CLIENT]
$
kali@kali: ~/CS433/Assignment1/SERVER
File Actions Edit View Help
Finish Time: 0.037084904
CPU Time: 0.0008171650000000016
(kali㉿kali)-[~/CS433/Assignment1/SERVER]
$ python server_code.py
[RUNNING] Server running at IP = 10.0.2.15 and port number = 5050
5
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at
('10.0.2.15', 45246)
packet received at server
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client
[CLOSED] server connection closed
Start Time: 0.031998123
Finish Time: 0.032690958
CPU Time: 0.0006928349999999958
(kali㉿kali)-[~/CS433/Assignment1/SERVER]
$


"Node: h1"
(root kali)-[/home/kali/CS433/Assignment1/SERVER]
python server_code.py
[RUNNING] Server running at IP = 10.0.0.1 and port number = 50505
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at ('10.0.0.2', 46720)
packet received at server
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client
[CLOSED] server connection closed
Start Time: 0.037074261
Finish Time: 0.038608606
CPU Time: 0.0015343449999999995
(root kali)-[/home/kali/CS433/Assignment1/SERVER]
"Node: h2"
(root kali)-[/home/kali/CS433/Assignment1/CLIENT]
python client_code.py
[RUNNING] Client started running
IP address of the host - 10.0.0.1
Port no of the host - 50505
[CREATED] clinet socket successfully created
[REQUESTING] client requesting for connection
User command - DWD a.txt
[SENDING] DWD commnad to Server
Sending to the server
a.txt downloaded successfully
[CLOSED] Clinet connection closed
Start Time: 0.037791275
Finish Time: 0.038636807
CPU Time: 0.000845532000000003
(root kali)-[/home/kali/CS433/Assignment1/CLIENT]


"Node: h1"
packet received at server
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client
[CLOSED] server connection closed
Start Time: 0.037074261
Finish Time: 0.038608606
CPU Time: 0.0015343449999999995
(root kali)-[/home/kali/CS433/Assignment1/SERVER]
python server_code.py
[RUNNING] Server running at IP = 10.0.0.1 and port number = 50505
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at ('10.0.0.2', 60512)
packet received at server
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client
[CLOSED] server connection closed
Start Time: 0.040564188
Finish Time: 0.041508096
CPU Time: 0.0009439080000000002
(root kali)-[/home/kali/CS433/Assignment1/SERVER]
#
"Node: h2"
a.txt downloaded successfully
[CLOSED] Clinet connection closed
Start Time: 0.037791275
Finish Time: 0.038636807
CPU Time: 0.00084553200000003
(root kali)-[/home/kali/CS433/Assignment1/CLIENT]
python client_code.py
[RUNNING] Client started running
IP address of the host - 10.0.0.1
Port no of the host - 50505
[CREATED] clinet socket successfully created
[REQUESTING] client requesting for connection
User command - DWD a.txt
[SENDING] DWD commnad to Server
Sending to the server
a.txt downloaded successfully
[CLOSED] Clinet connection closed
Start Time: 0.033250529
Finish Time: 0.034512376
CPU Time: 0.0012618469999999965
(root kali)-[/home/kali/CS433/Assignment1/CLIENT]
#



in ms	Mininet	Virtual Machine
1st	0.8	2.5
2nd	1.26	1.6
3rd	1.37	1.4
Median	1.26	1.6

For performance I did just DWD command on VM and mininet. I import the time library and them calculate the time difference like we did in assignment 1 of computer architecture.

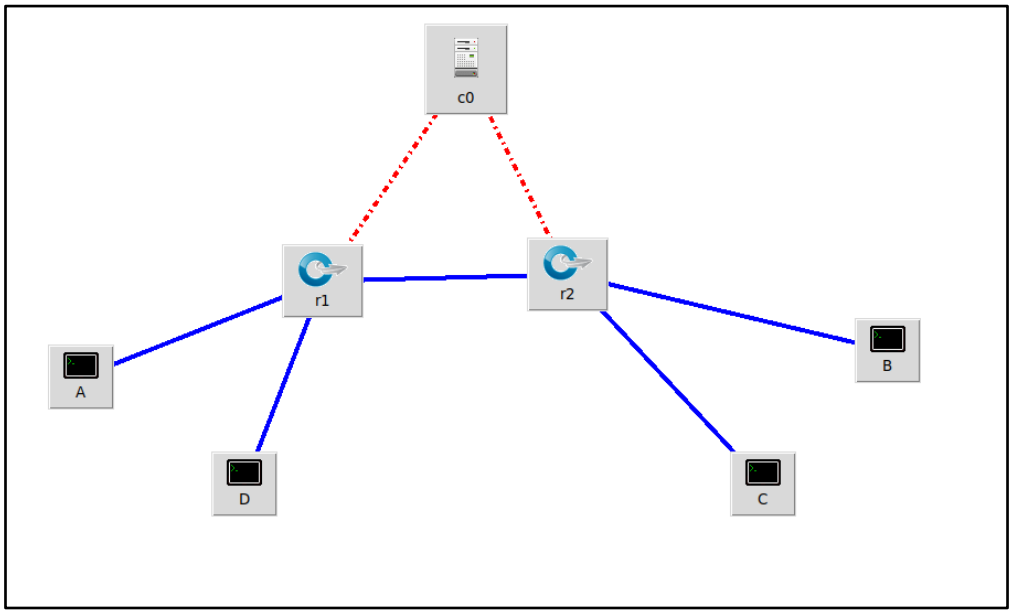
In mininet it took 1.26 ms and 1.6 ms in virtual machine.

- From the looks Mininet is faster than virtual machine in running the client server model.

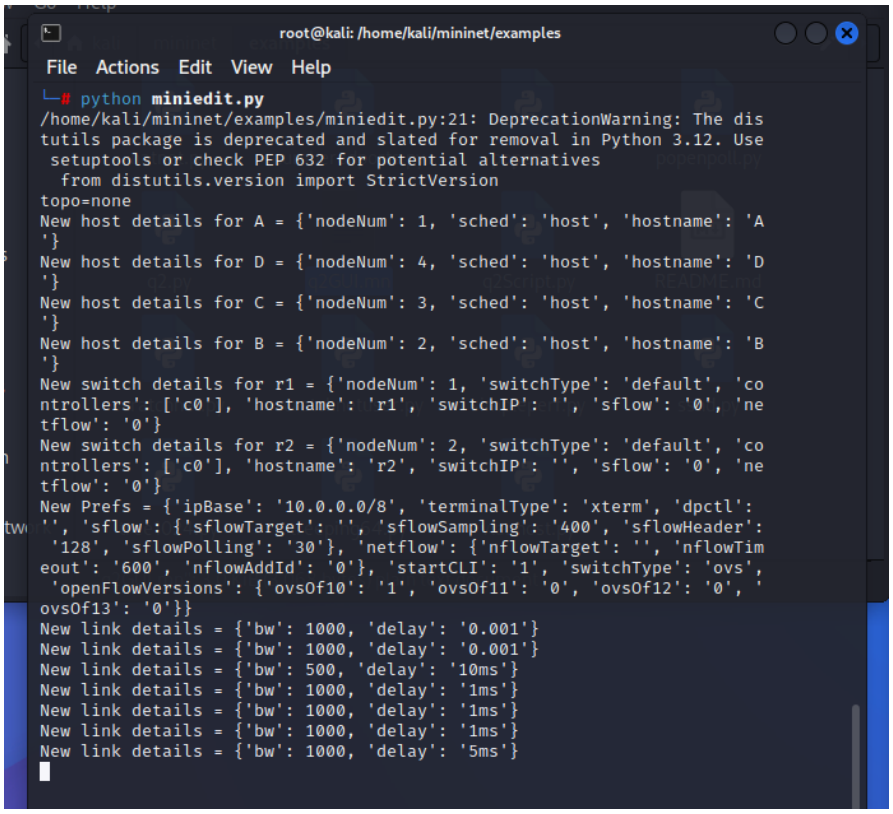
Question 2

(A) Implementation:

For making the given topologies I use the **miniedit** which is a graphical way of making the topologies. I first make the topology and set the link parameters then generated the python code for that gui and then ran that python api file to get mininet for the given topology.




```
def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)

    info( '*** Add switches\n')
    r1 = net.addSwitch('r1', cls=OVSKernelSwitch)
    r2 = net.addSwitch('r2', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    A = net.addHost('A', cls=Host, ip='10.0.0.1', defaultRoute=None)
    B = net.addHost('B', cls=Host, ip='10.0.0.2', defaultRoute=None)
    C = net.addHost('C', cls=Host, ip='10.0.0.3', defaultRoute=None)
    D = net.addHost('D', cls=Host, ip='10.0.0.4', defaultRoute=None)

    info( '*** Add links\n')
    Ar1 = {'bw':1000,'delay':'1ms'}
    net.addLink(A, r1, cls=TCLink , **Ar1)
    r1D = {'bw':1000,'delay':'1ms'}
    net.addLink(r1, D, cls=TCLink , **r1D)
    r1r2 = {'bw':500,'delay':'10ms'}
    net.addLink(r1, r2, cls=TCLink , **r1r2)
    r2C = {'bw':1000,'delay':'5ms'}
    net.addLink(r2, C, cls=TCLink , **r2C)
    r2B = {'bw':1000,'delay':'1ms'}
    net.addLink(r2, B, cls=TCLink , **r2B)
```



Then run the *python q2Script.py* file containing the code for the topology

```
┌──(root㉿kali)-[/home/kali/mininet/examples]
└─# python q2Script.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (1000.
00Mbit 1ms delay) (500.00Mbit 10ms delay) (500.00Mbit 10ms delay) (1000.00Mbit
 5ms delay) (1000.00Mbit 5ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms d
elay) *** Starting network
*** Configuring hosts
A B C D
*** Starting controllers
*** Starting switches
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (500.00Mbit 10ms delay) (500.0
0Mbit 10ms delay) (1000.00Mbit 5ms delay) (1000.00Mbit 1ms delay) *** Post con
figure switches and hosts
*** Starting CLI:
mininet> nodes
available nodes are:
A B C D c0 r1 r2
mininet> net
A A-eth0:r1-eth1
B B-eth0:r2-eth3
C C-eth0:r2-eth2
D D-eth0:r1-eth2
r1 lo:  r1-eth1:A-eth0 r1-eth2:D-eth0 r1-eth3:r2-eth1
r2 lo:  r2-eth1:r1-eth3 r2-eth2:C-eth0 r2-eth3:B-eth0
c0
mininet> dump
<Host A: A-eth0:10.0.0.1 pid=23853>
<Host B: B-eth0:10.0.0.2 pid=23855>
<Host C: C-eth0:10.0.0.3 pid=23857>
<Host D: D-eth0:10.0.0.4 pid=23859>
<OVSSwitch r1: lo:127.0.0.1,r1-eth1:None,r1-eth2:None,r1-eth3:None pid=23845>
<OVSSwitch r2: lo:127.0.0.1,r2-eth1:None,r2-eth2:None,r2-eth3:None pid=23848>
<Controller c0: 127.0.0.1:6633 pid=23837>
mininet>
```

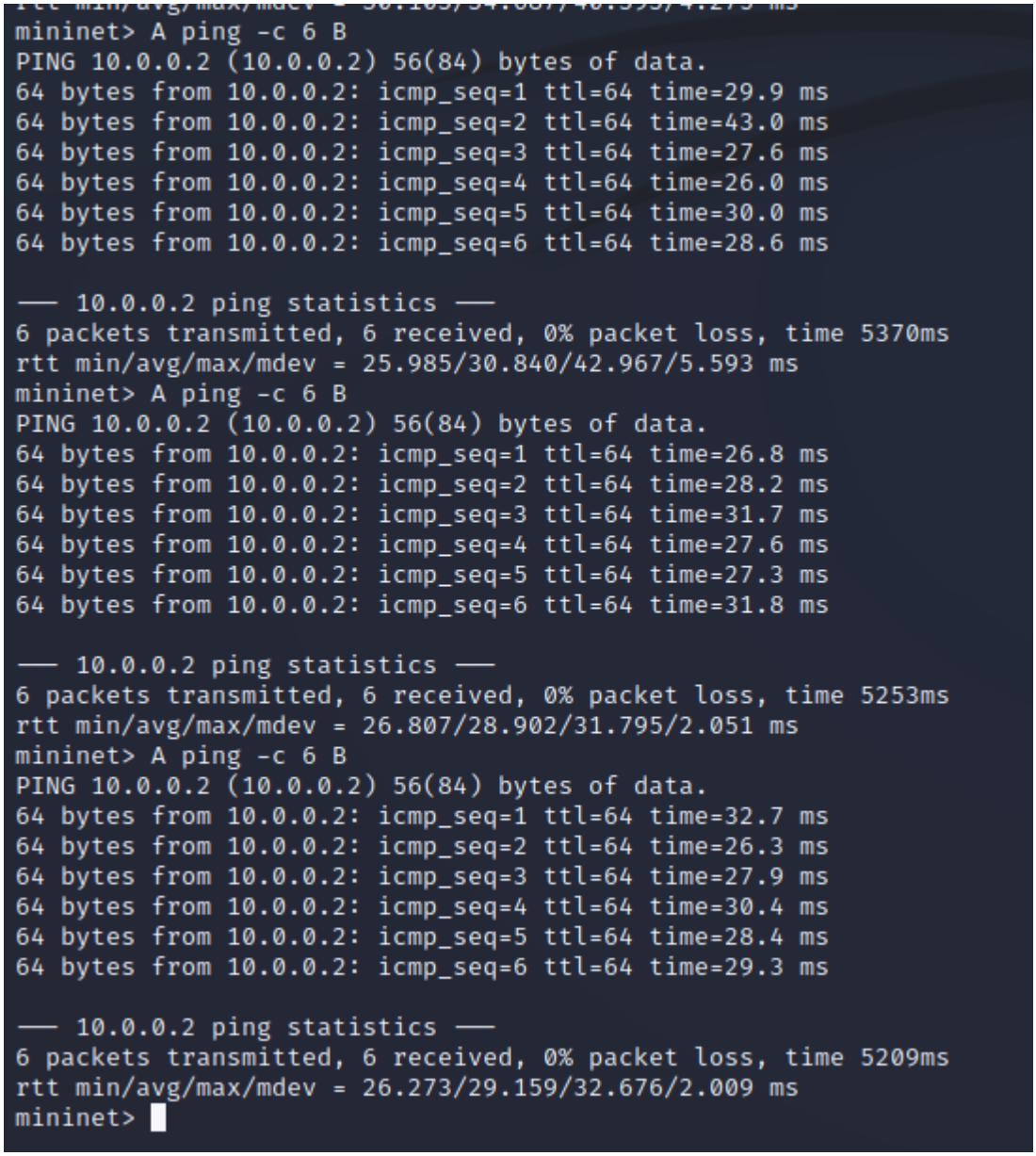
Checking whether our topology is working, *pingall*

```
<OVSSwitch r1: lo:127.0.0.1,r1-eth1:None,r1-eth2:None,r1-eth3:None pid=23845>
<OVSSwitch r2: lo:127.0.0.1,r2-eth1:None,r2-eth2:None,r2-eth3:None pid=23848>
<Controller c0: 127.0.0.1:6633 pid=23837>
mininet> pingall
*** Ping: testing ping reachability
A → B C D
B → A C D
C → A B D
D → A B C
*** Results: 0% dropped (12/12 received)
mininet>
```

(B) Latency:

Measured by *A ping − c 6 B* (Stopping after transferring 6 packets)

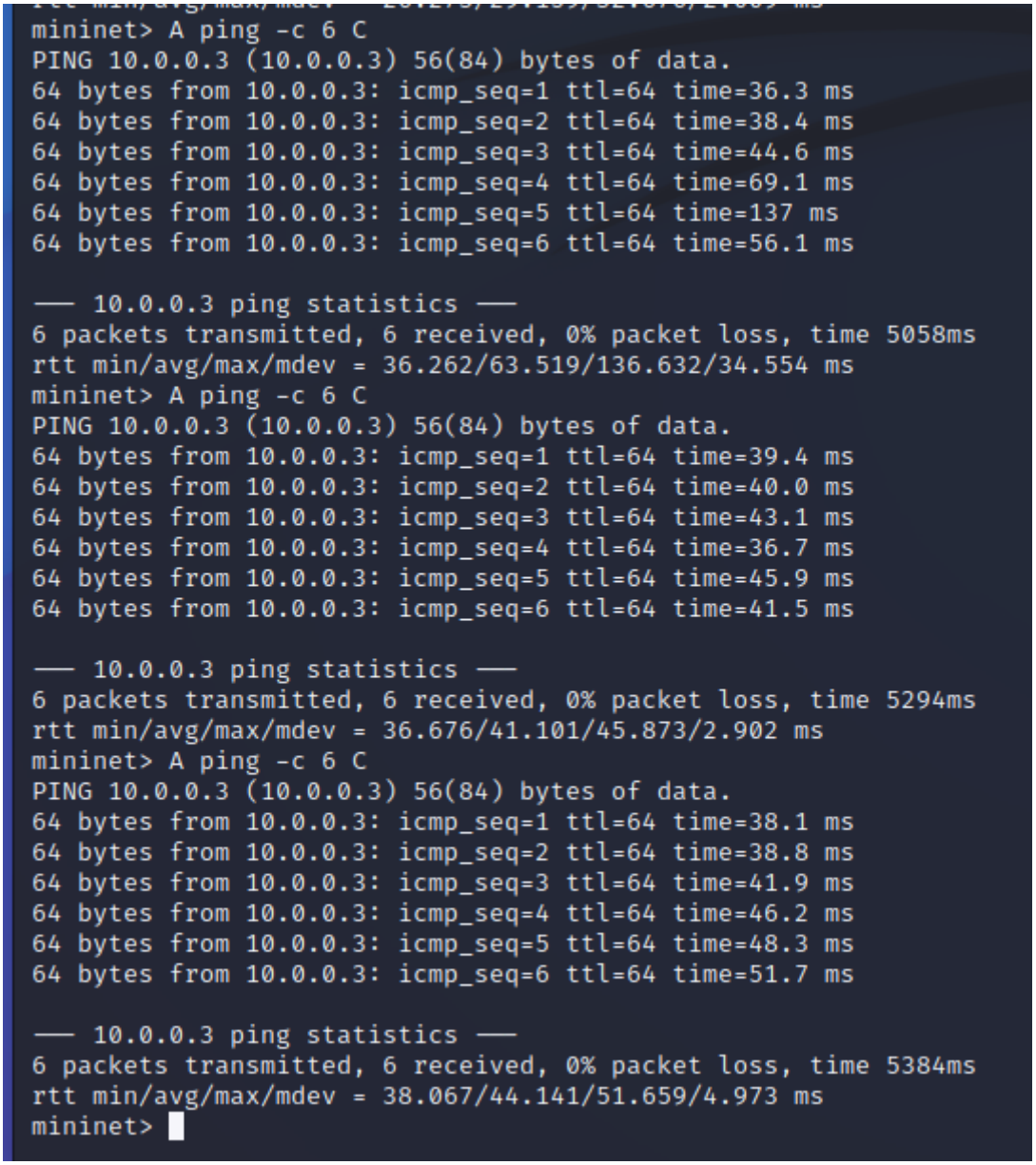
AB:



	RTT
1st Iteration	30.84
2nd Iteration	28.902
3rd Iteration	29.159
Average	**29.63366667**

The average the RTT for the node pairs AB is 29.63ms

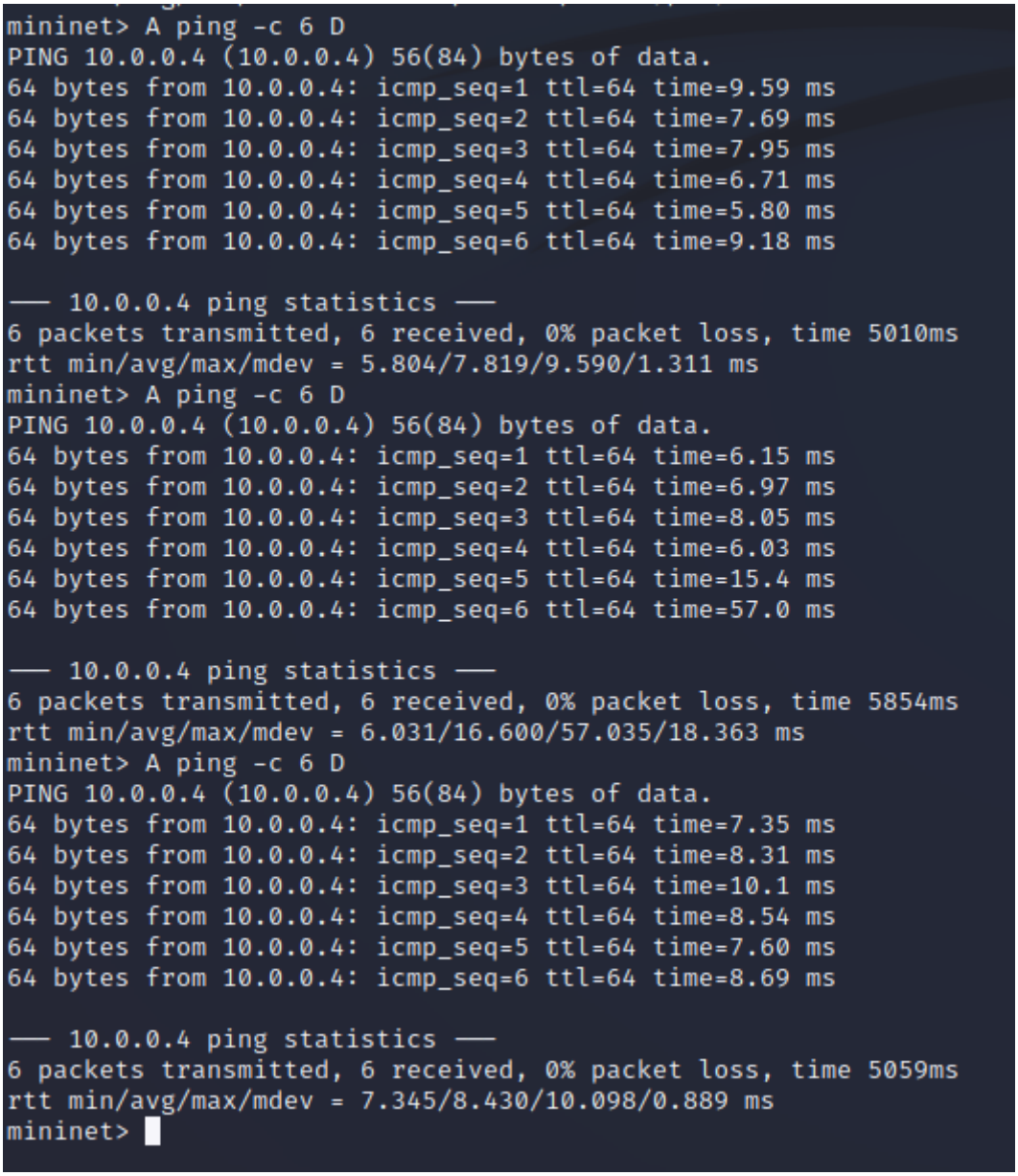
AC:



	RTT
1st Iteration	63.519
2nd Iteration	41.101
3rd Iteration	44.141
Average	**49.587**

The average the RTT for the node pairs AB is 49.58ms

AD:



	RTT
1st Iteration	7.819
2nd Iteration	16.6
3rd Iteration	8.43
Average	**10.94966667**

The average the RTT for the node pairs AB is 10.95ms

BC:

```
mininet> B ping -c 6 C
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=19.9 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=18.4 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=26.9 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=60.1 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=17.7 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=16.2 ms

── 10.0.0.3 ping statistics ──
6 packets transmitted, 6 received, 0% packet loss, time 5031ms
rtt min/avg/max/mdev = 16.238/26.520/60.059/15.379 ms
mininet> B ping -c 6 C
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=17.2 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=106 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=35.0 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=28.9 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=15.6 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=15.5 ms

── 10.0.0.3 ping statistics ──
6 packets transmitted, 6 received, 0% packet loss, time 5105ms
rtt min/avg/max/mdev = 15.472/36.409/106.308/32.105 ms
mininet> B ping -c 6 C
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=15.6 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=14.7 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=13.7 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=16.5 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=15.4 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=16.4 ms

── 10.0.0.3 ping statistics ──
6 packets transmitted, 6 received, 0% packet loss, time 5211ms
rtt min/avg/max/mdev = 13.683/15.378/16.518/0.983 ms
mininet>
```

	RTT
1st Iteration	26.52
2nd Iteration	36.409
3rd Iteration	15.378
Average	**26.10233333**

The average the RTT for the node pairs AB is 26.10ms

BD:

```
rtt min/avg/max/mdev = 15.683/15.378/16.518/0.983 ms
mininet> B ping -c 6 D
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=32.9 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=31.3 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=154 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=47.9 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=26.7 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=32.9 ms

--- 10.0.0.4 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5091ms
rtt min/avg/max/mdev = 26.651/54.232/153.766/44.992 ms
mininet> B ping -c 6 D
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=30.6 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=38.8 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=33.1 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=30.3 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=45.7 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=39.9 ms

--- 10.0.0.4 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5613ms
rtt min/avg/max/mdev = 30.327/36.403/45.720/5.570 ms
mininet> B ping -c 6 D
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=30.9 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=29.8 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=33.7 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=28.7 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=28.0 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=35.7 ms

--- 10.0.0.4 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5094ms
rtt min/avg/max/mdev = 28.045/31.147/35.655/2.714 ms
mininet>
```

	RTT
1st Iteration	54.232
2nd Iteration	36.403
3rd Iteration	31.147
Average	**40.594**

The average the RTT for the node pairs AB is 40.594ms

(C) Throughput

Before plotting the result, I have stored the result of server at host A in ra file. Now we are required to do some post processing in the file to make the data available for plotting. It is stored in the file named nr.

$cat\ ra \mid grep\ sec \mid head\ -15 \mid tr\ -\ "\ " \mid awk\ '\{print\ \$4, \$8\}'\ >\ nr$

Used the GNU plots for plotting

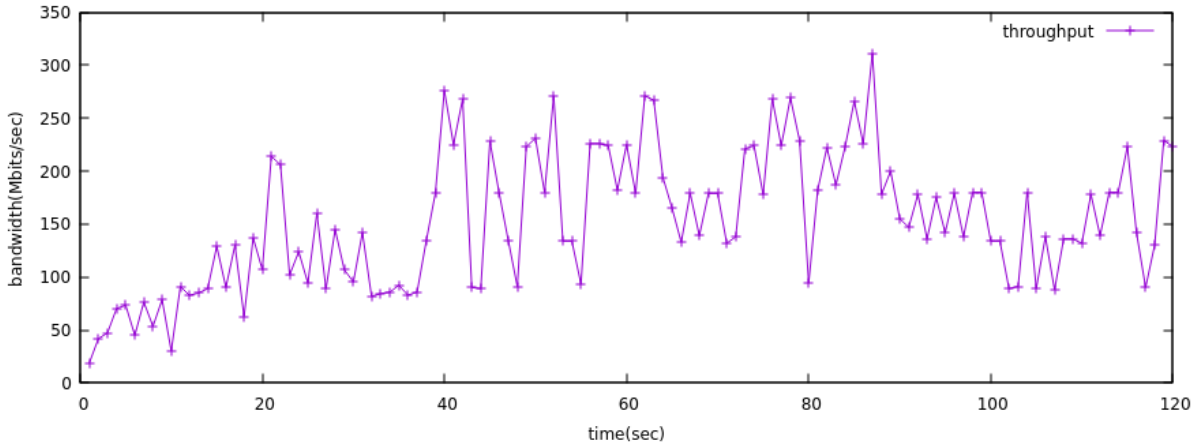
$sudo\ apt - get\ install\ gnuplot$

Steps used to make the plot

$gnuplot$ $plot\ "nr"\ title\ "throughput"\ with\ linespoints$ $set\ xlabel\ "time(sec)"$ $set\ ylabel\ "bandwidth(Mbits/sec)"$ $replot$ The plotted graph

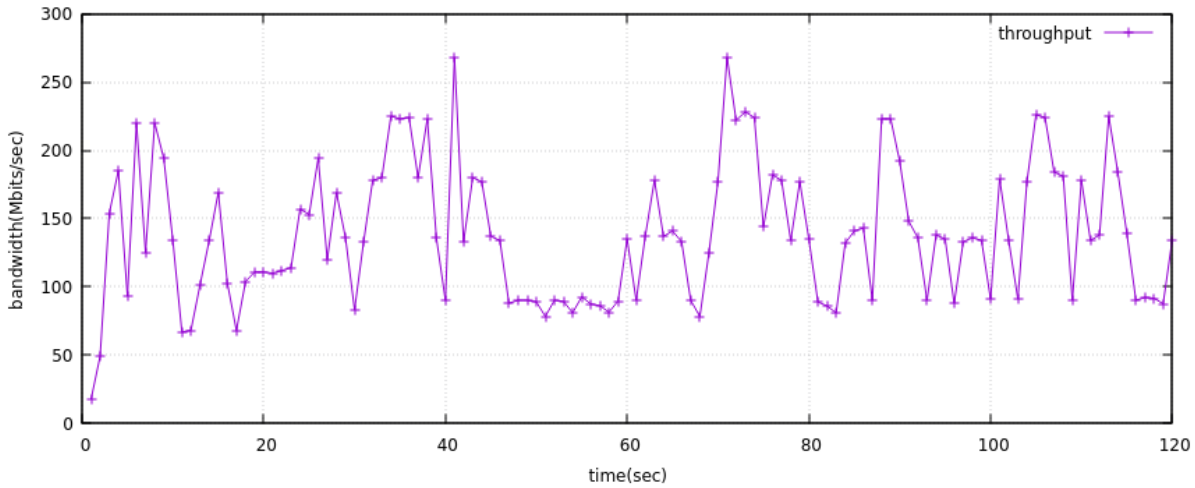
1st Iteration:

At client B:



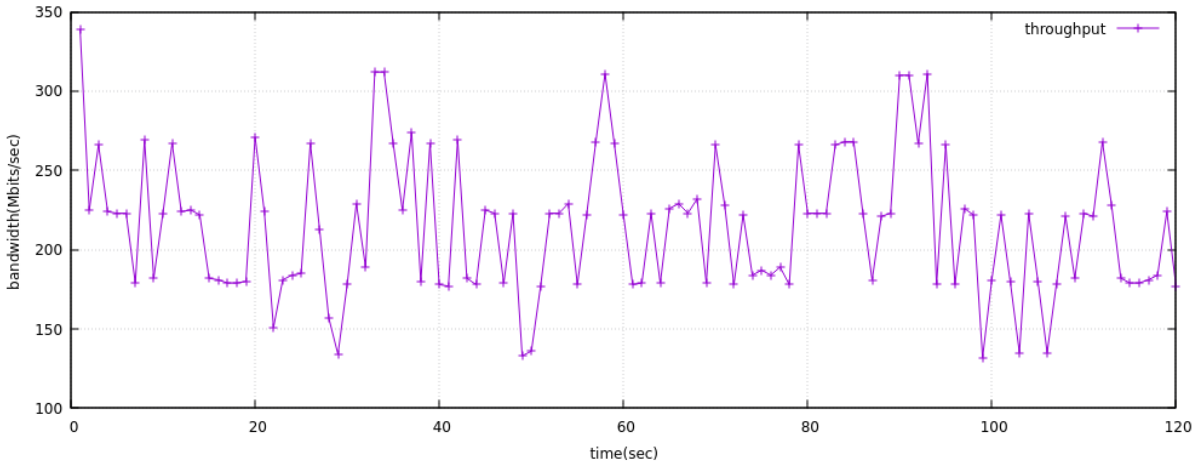
2nd Iteration:

At client B:



3rd Iteration:

At client B:



All data in Mbit/sec

	B (Client)	A (Server)	Transfer (in GB)
1st Iteration	150	150	2.03
2nd Iteration	138	138	1.94
3rd Iteration	214	214	3
Average	**167.3333333**	**167.3333333**	**2.323333333**

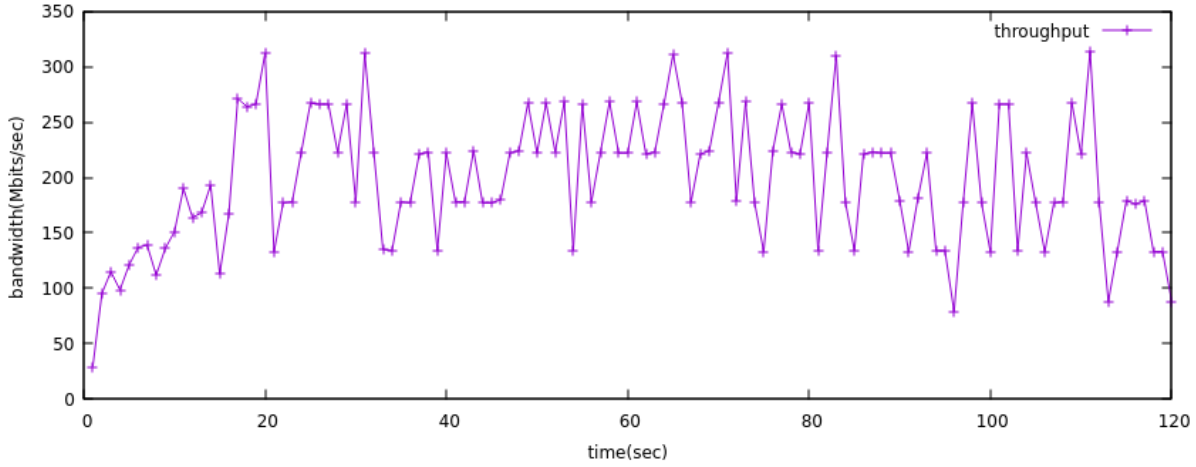
The average throughput at client B is 167.33 Mbit/sec when server transfer the data of 2.32 GB.

It is very small when compared with the bottle neck of the path which is (500Mbit/sec)

(D) **Delay Impact:** Server at host A and client at the host C, now doing the Iperf

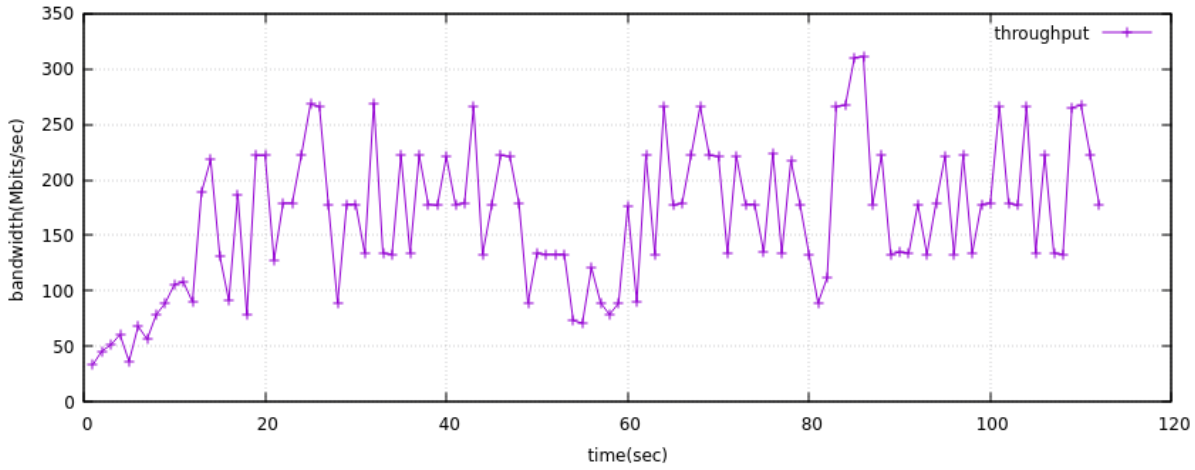
1st Iteration:

At client C:



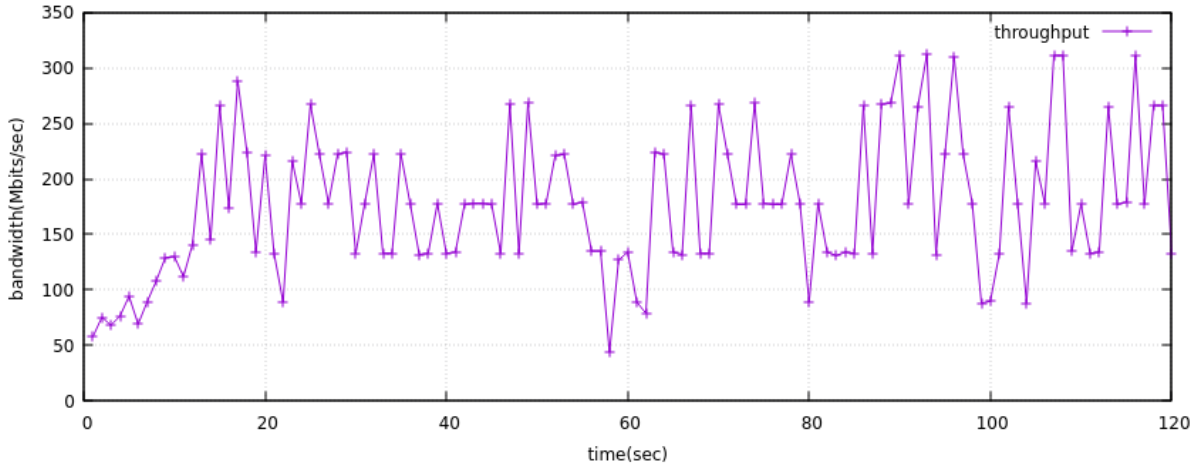
2nd Iteration:

At client C:



3rd Iteration:

At client C:



All data in Mbit/sec

	C (Client)	A (Server)	Transfer (in GB)
1st Iteration	197	198	2.77
2nd Iteration	166	166	2.34
3rd Iteration	176	176	2.47
Average	**179.6666667**	**180**	**2.526666667**

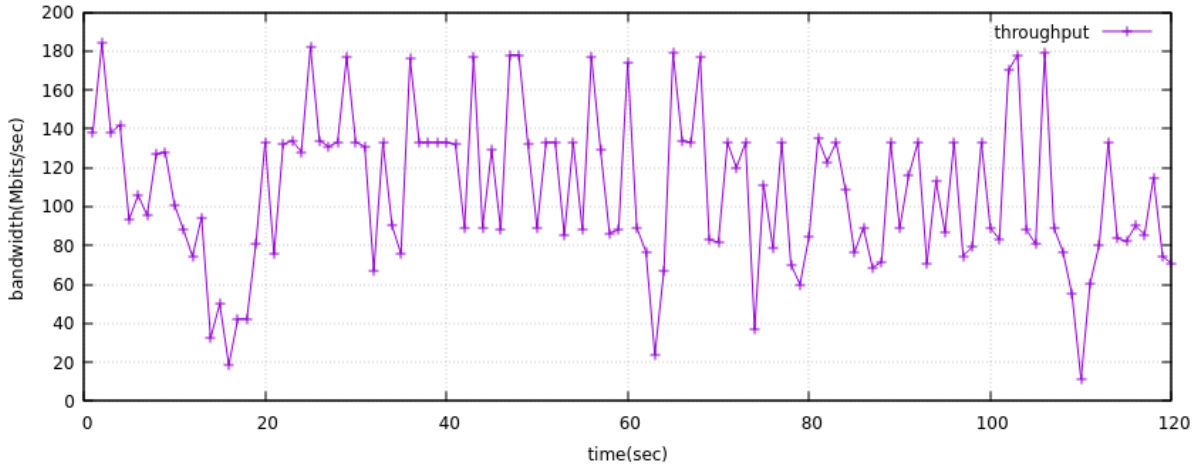
The average through put at C is 179.67 Mbit/sec when the server at host A transfer the 2.52GB of data.

It is very small when compared with the bottle neck of the path which is (500Mbit/sec)

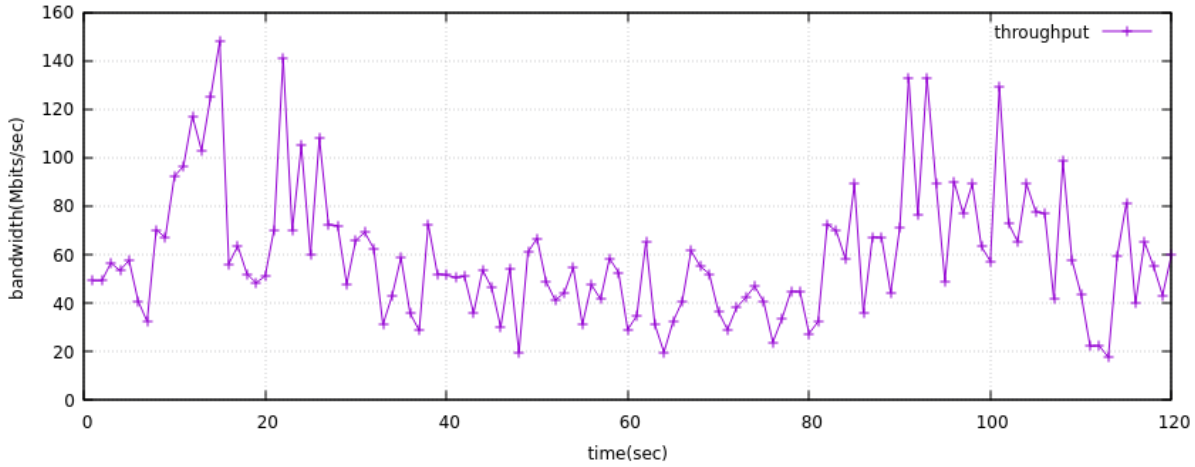
(E) Concurrency(i): Server at host A and client at the host B, host C (running simultaneously) and then doing the $iperf$

1st Iteration:

At client B:

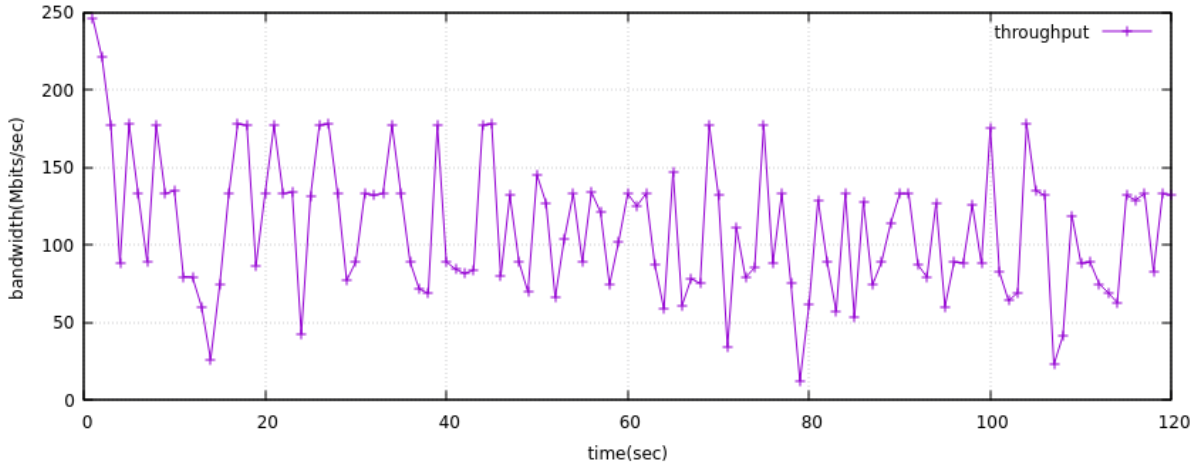


At client C:

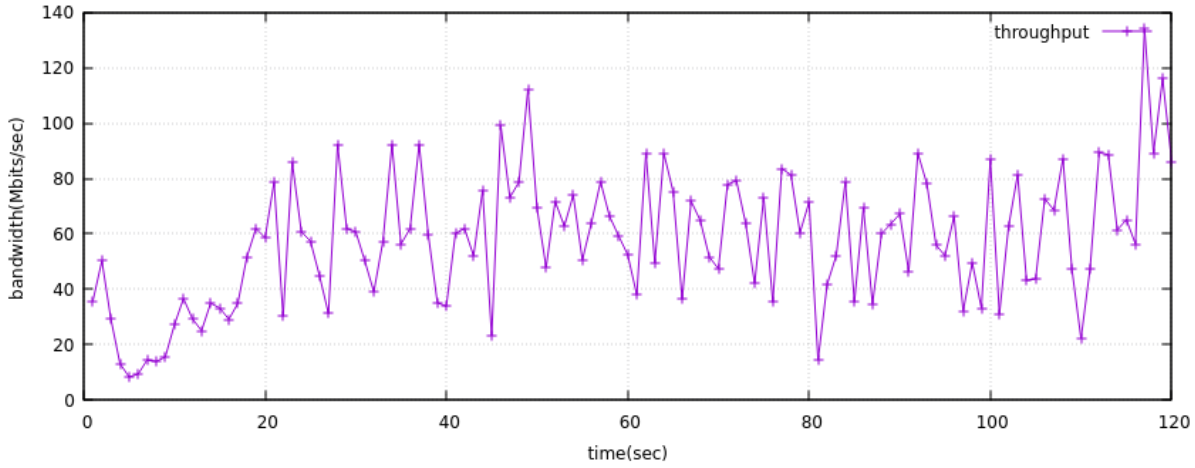


2nd Iteration:

At client B:

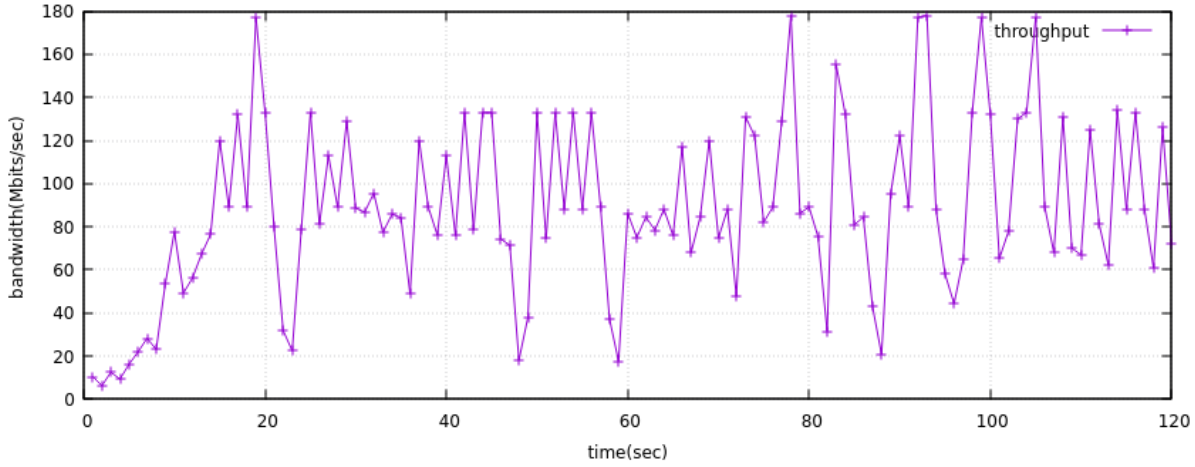


At Client C:

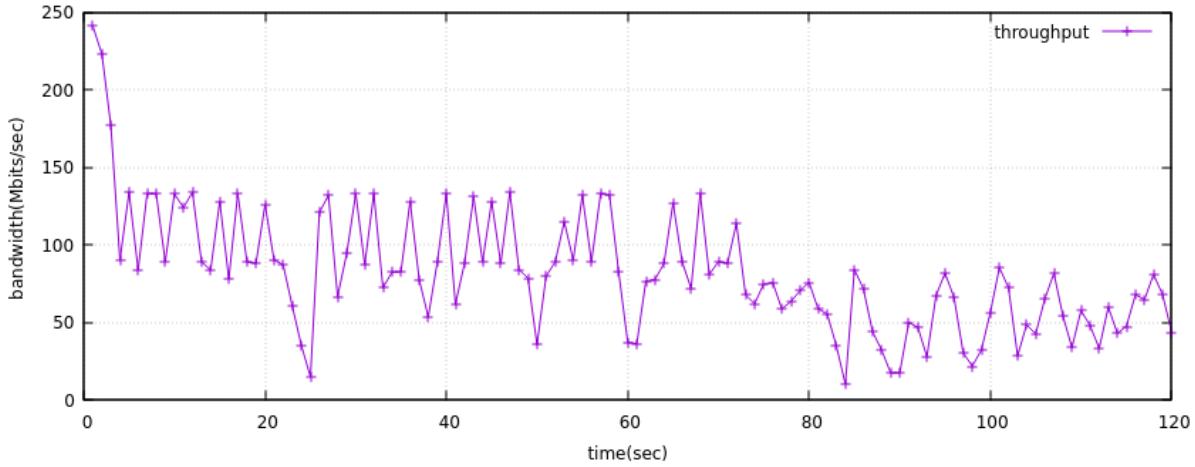


3rd Iteration:

At client B:



At client C:



	B (Client)	C (Client)	A (Server)	Transfer at B (GB)	Transfer at C (GB)
1st Iteration	107	58.4	107	1.51	0.883
2nd Iteration	110	57.1	110	1.54	0.882
3rd Iteration	87.4	80.8	87.6	1.24	1.15
Average	**101.4666667**	**65.43333333**	**101.5333333**	**1.43**	**0.9716666667**

The average through put at B is 101.47 Mbit/sec when the server at host A transfer the 1.43 GB of data.

The average through put at C is 65.433 Mbit/sec when the server at host A transfer the 0.97 GB of data.

It is very small when compared with the bottle neck of the path which is (500Mbit/sec)

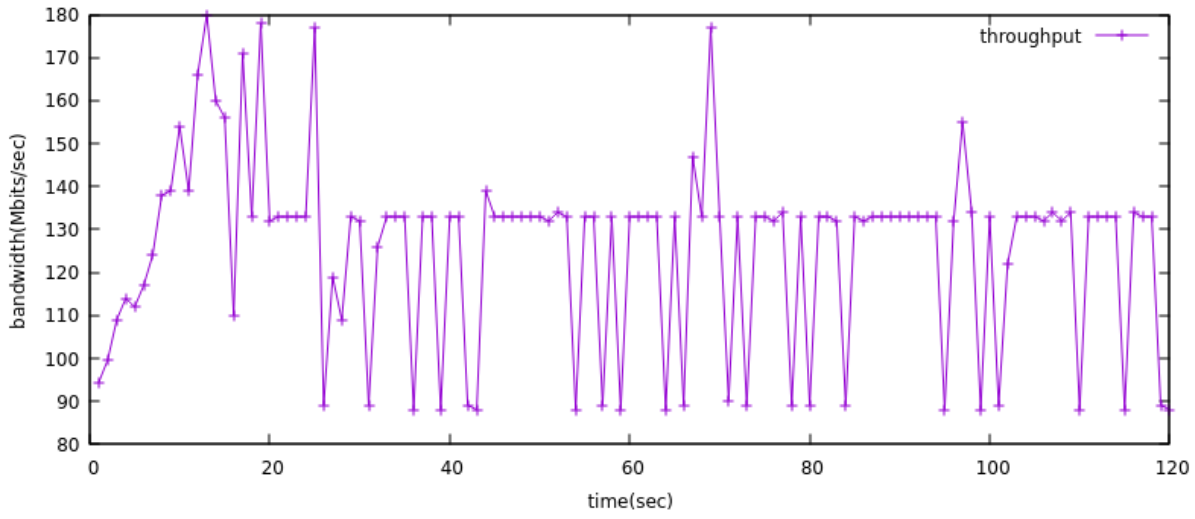
(F) Fairness(ii):

Changed the Link 4 delay to 1ms. Then again run the code. Server at A and clients at B, C and running them simultaneously.

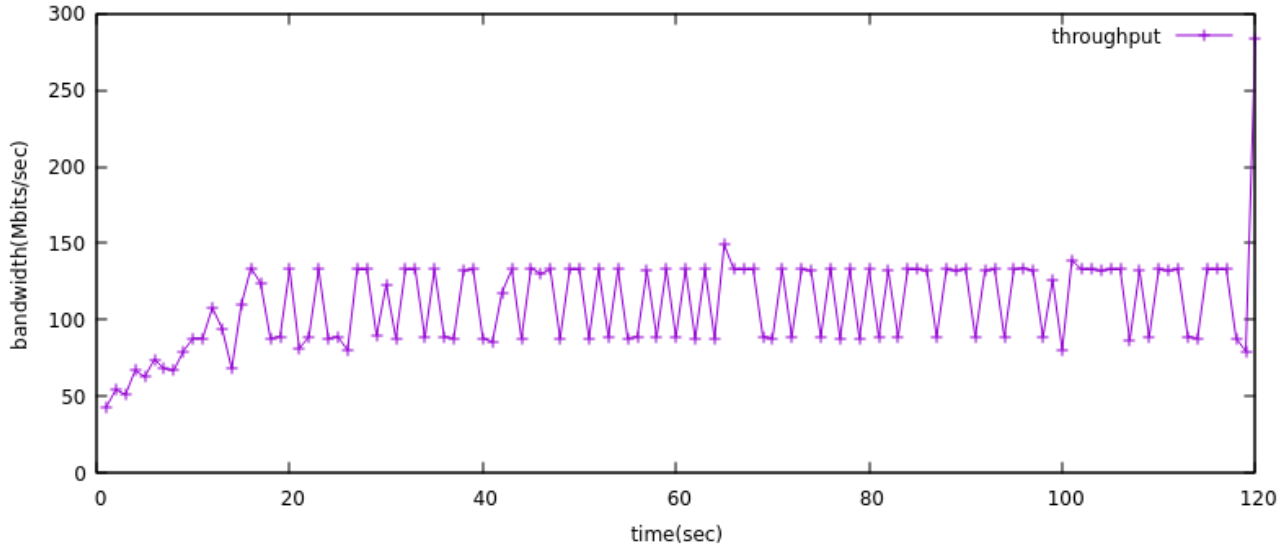
```
┌──(root㉿kali)-[/home/kali/mininet/examples]
└─# python miniedit.py
/home/kali/mininet/examples/miniedit.py:21: DeprecationWarning: The d
istutils package is deprecated and slated for removal in Python 3.12.
 Use setuptools or check PEP 632 for potential alternatives
  from distutils.version import StrictVersion
topo=none
New link details = {'bw': 1000, 'delay': '1ms'}
Getting Hosts and Switches.
Getting controller selection:ref
Getting Links.
(1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms dela
y) (1000.00Mbit 1ms delay) (500.00Mbit 10ms delay) (500.00Mbit 10ms d
elay) (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit 1m
s delay) (1000.00Mbit 1ms delay) *** Configuring hosts
A B C D
**** Starting 1 controllers
c0
**** Starting 2 switches
r1 (1000.00Mbit 1ms delay) (1000.00Mbit 1ms delay) (500.00Mbit 10ms d
elay) r2 (500.00Mbit 10ms delay) (1000.00Mbit 1ms delay) (1000.00Mbit
 1ms delay)
No NetFlow targets specified.
No sFlow targets specified.


 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTT
ON. Not exiting will prevent MiniEdit from quitting and will prevent
you from starting the network again during this session.

*** Starting CLI:
mininet> net
A A-eth0:r1-eth1
B B-eth0:r2-eth3
C C-eth0:r2-eth2
D D-eth0:r1-eth2
r1 lo:  r1-eth1:A-eth0 r1-eth2:D-eth0 r1-eth3:r2-eth1
r2 lo:  r2-eth1:r1-eth3 r2-eth2:C-eth0 r2-eth3:B-eth0
c0
mininet>
```

1st Iteration:

At client B:

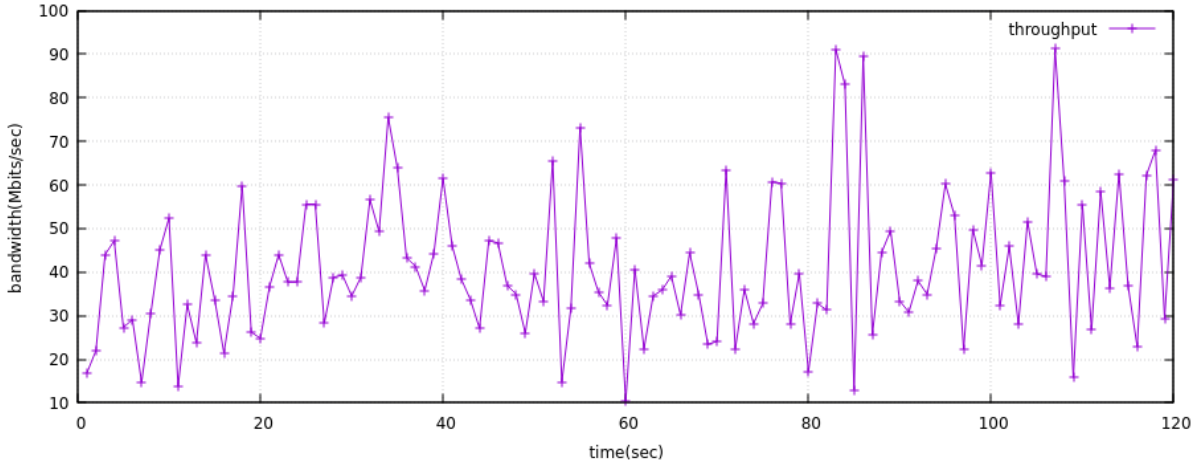


At client C:

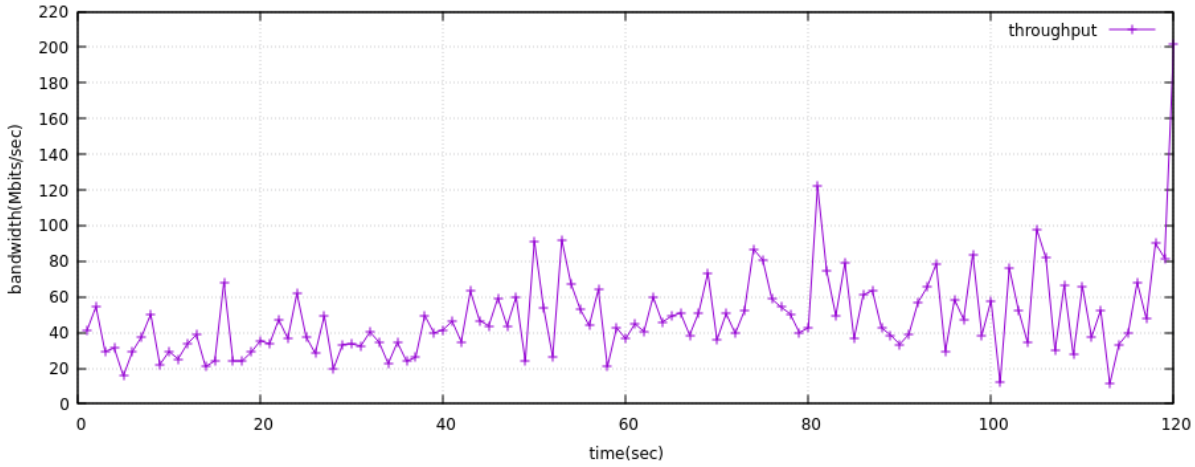


2nd Iteration:

At client B:

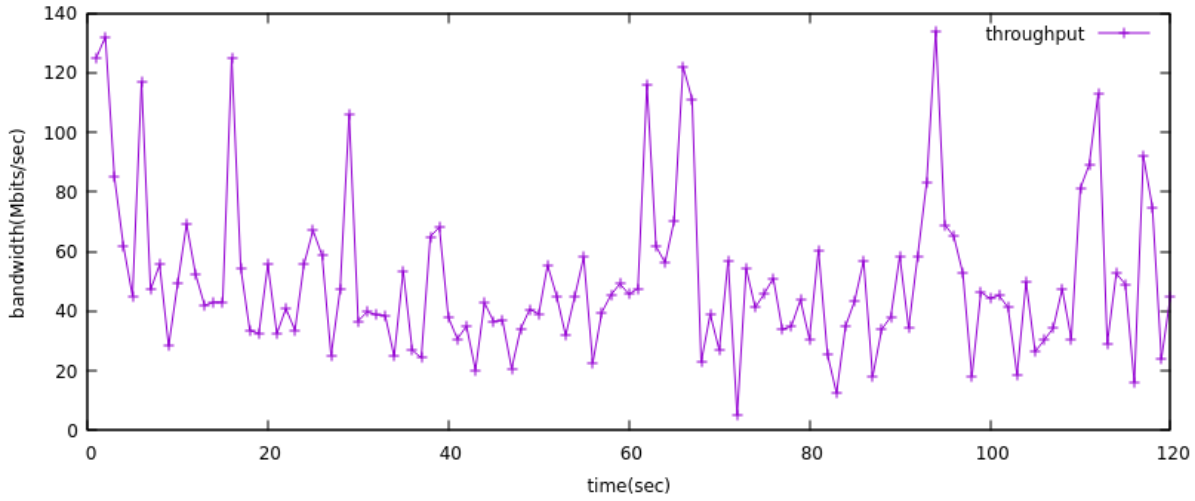


At Client C:

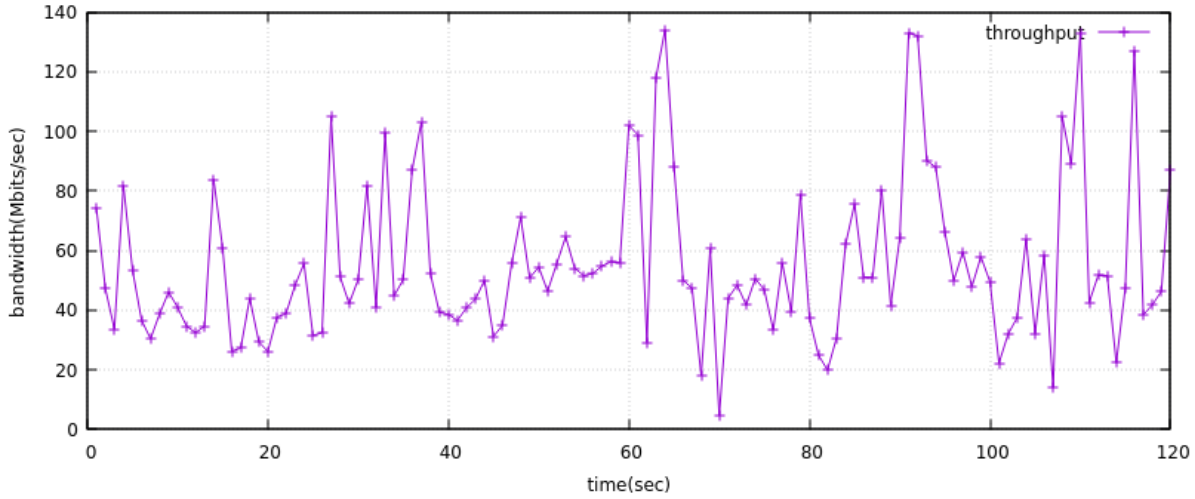


3rd Iteration:

At client B:



At Client C:



All data in Mbit/sec

	B (Client)	C (Client)	A (Server)
1st Iteration	125	110	110
2nd Iteration	40.4	48.3	48.3
3rd Iteration	49.6	190	54.7
Average	**71.66666667**	**116.1**	**71**

The average through put at B is 71.67 Mbit/sec when the server at host A.

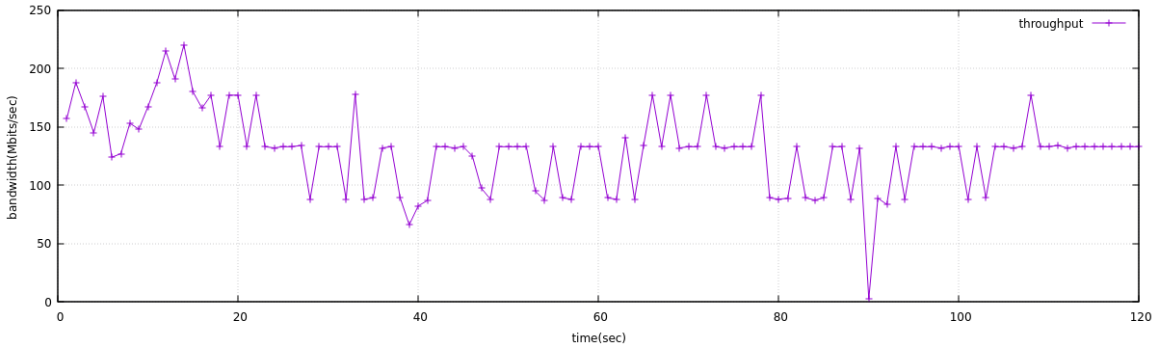
The average through put at C is 116.1 Mbit/sec when the server at host A.

- Still, It is very small when compared with the bottle neck of the path which is (500Mbit/sec)
- There is decrease in the throughput for client B but increase in the throughput for client C. (this is consistent as we have decreased the delay for link 4 thus its through put must increase.)

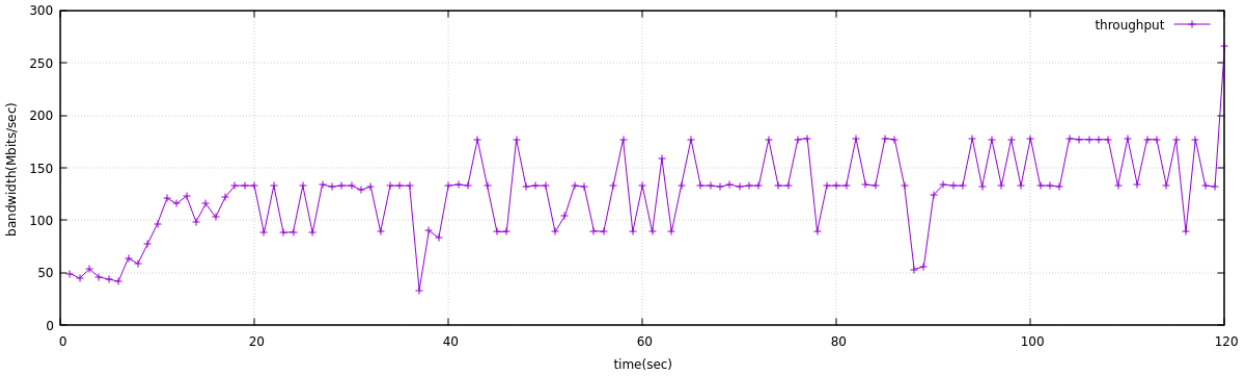
(E) Concurrency(iii):

1st Iteration:

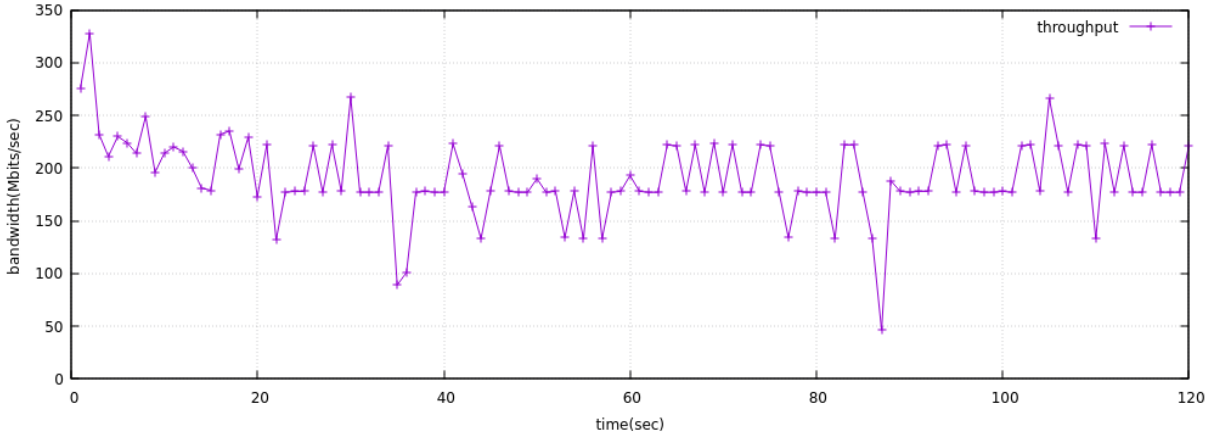
At client B:



At client C:

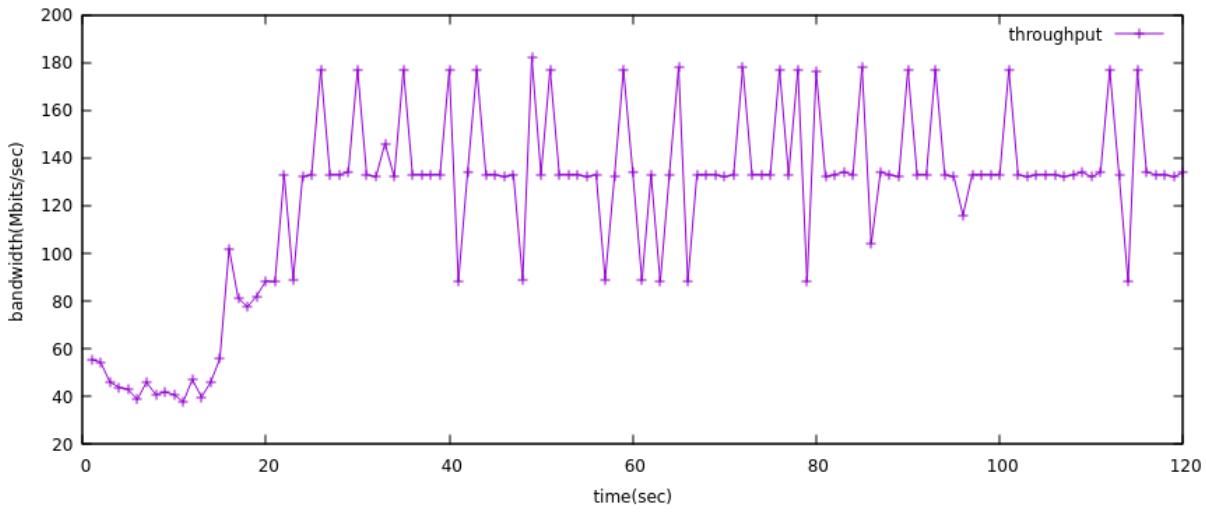


At Client D:

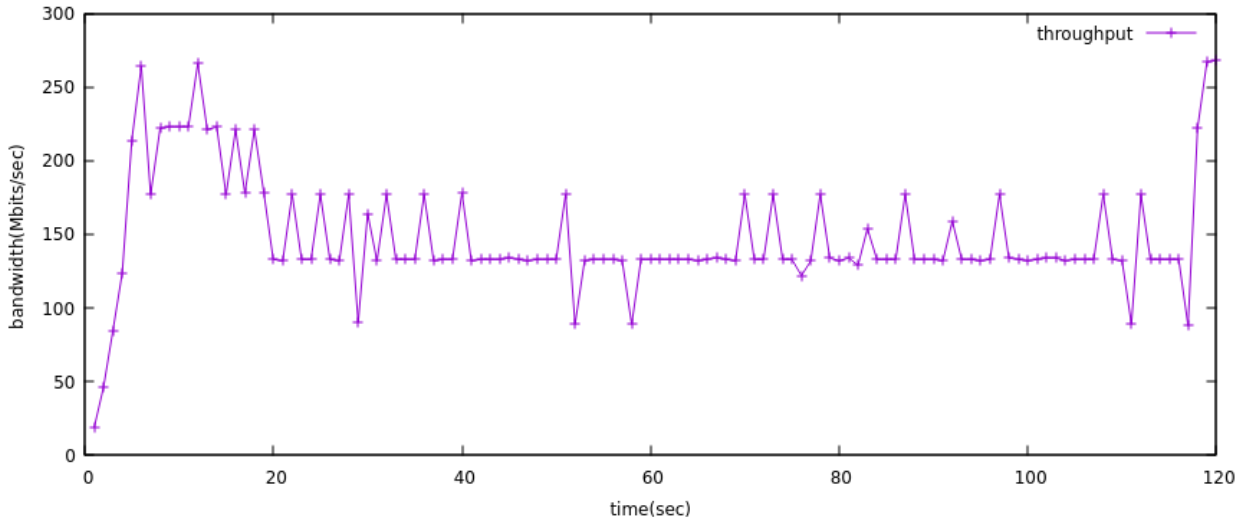


2nd Iteration:

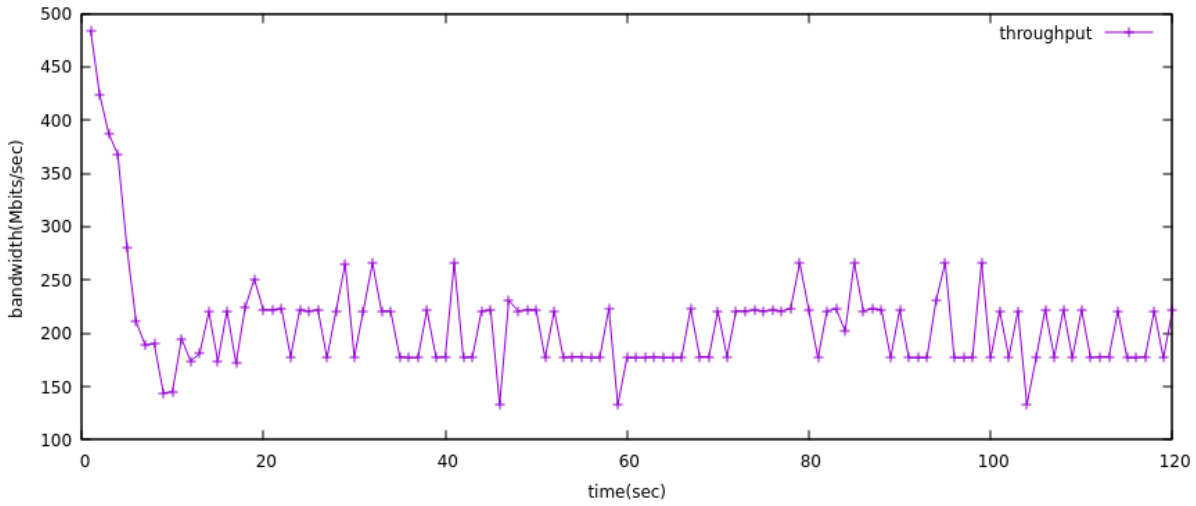
At client B:



At client C:

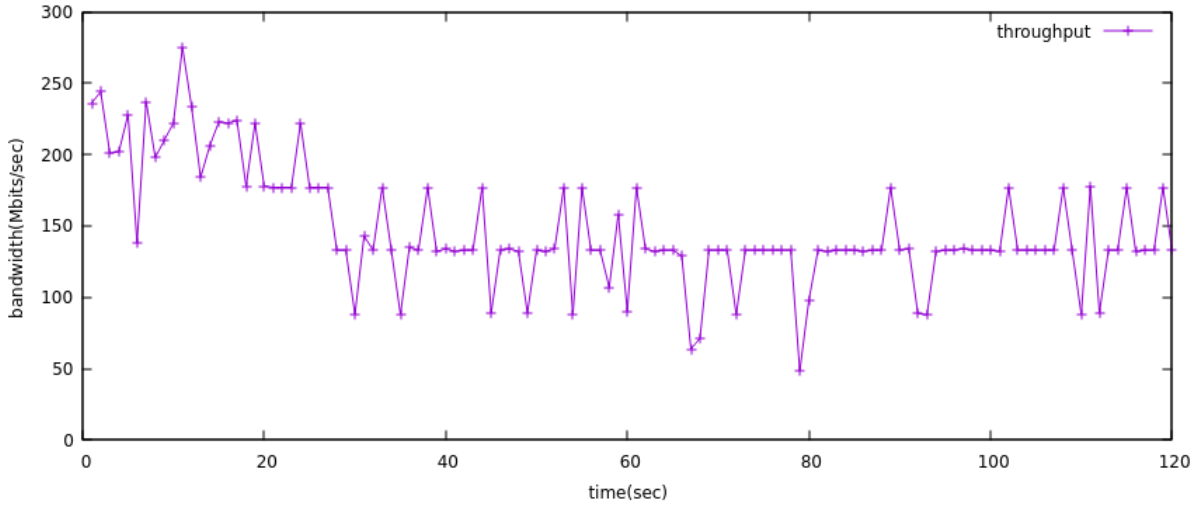


At client D:

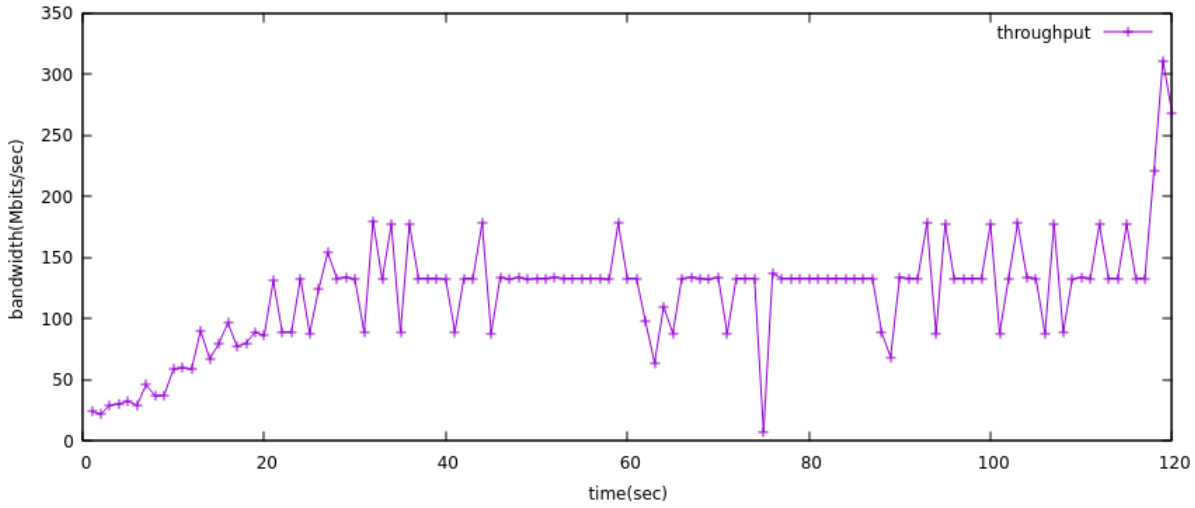


3rd Iteration:

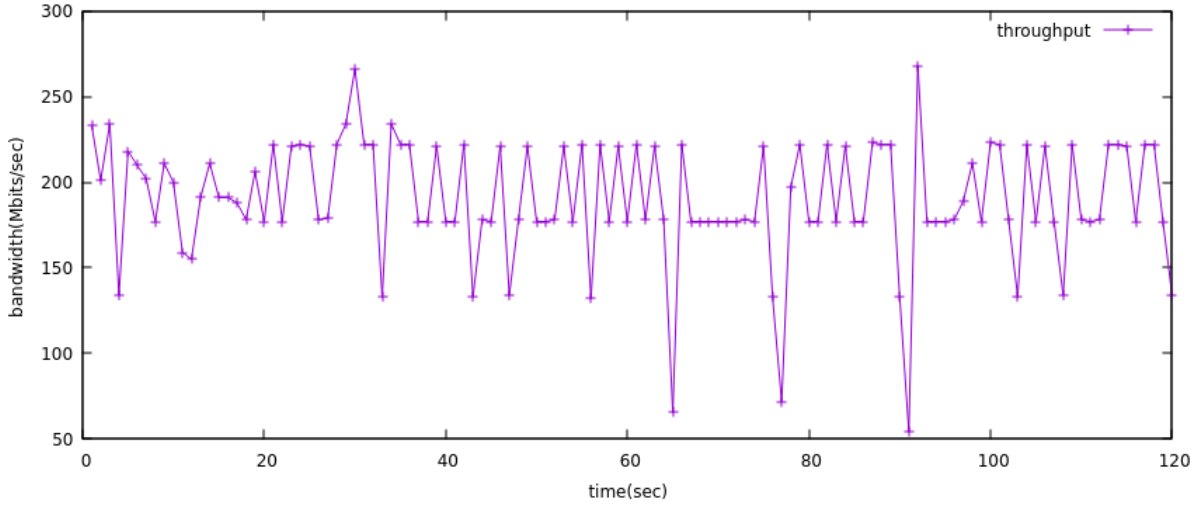
At client B:



At client C:



At client D:



All the data in Mbit/sec,

	B (Client)	C (Client)	D (Client)	A (Server)

1st Iteration	129	127	191	191
2nd Iteration	122	148	208	148
3rd Iteration	147	120	190	120
Average	**132.6666667**	**131.6666667**	**196.3333333**	**153**

The average through put at B is 132.66 Mbit/sec when the server at host A.

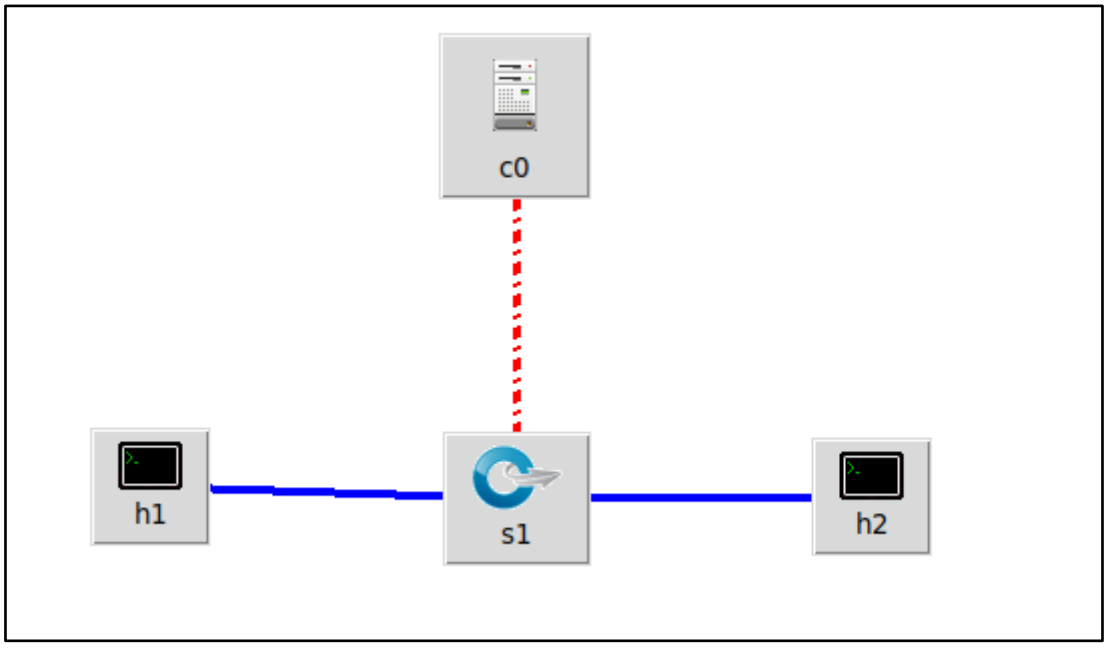
The average through put at C is 131.667 Mbit/sec when the server at host A.

The average through put at C is 196.33 Mbit/sec when the server at host A.

- Still, it is very small when compared with the bottle neck of the path which is (500Mbit/sec)
- There is increase in the throughput. (this is consistent as we have decreased the delay for link 4 thus it's through put must increase.)

Question 3:

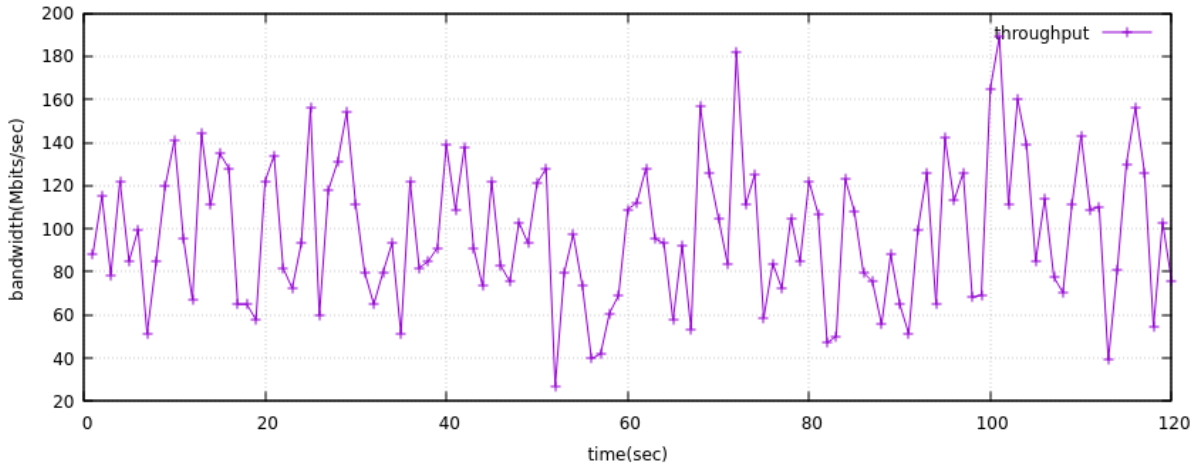
The topology is:



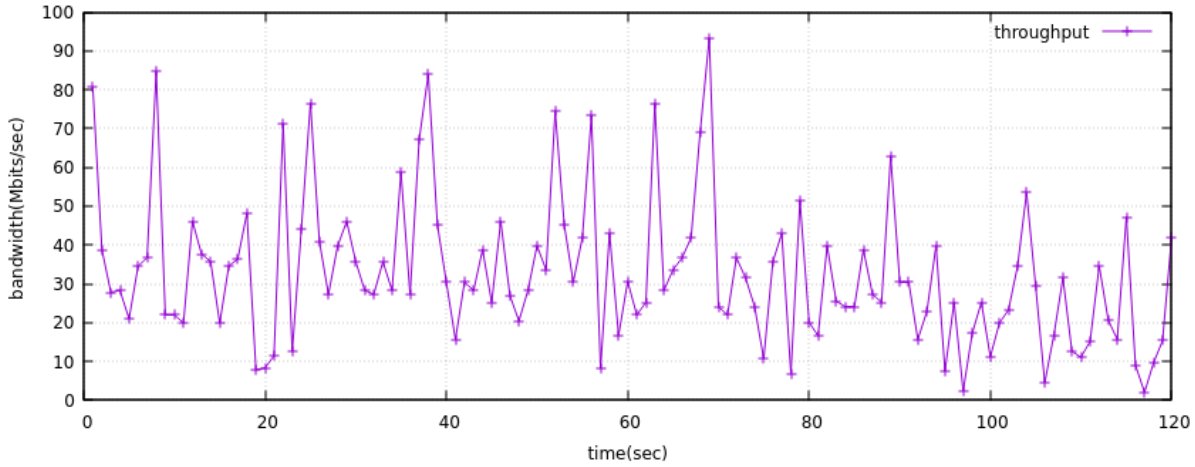
```python

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)

    info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)

    info( '*** Add links\n')
    h1s1 = {'bw':1000,'loss':1}
    net.addLink(h1, s1, cls=TCLink , **h1s1)
    s1h2 = {'bw':1000,'loss':1}
    net.addLink(s1, h2, cls=TCLink , **s1h2)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s1').start([c0])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()

```

```
[sudo] password for kali:
┌──(root㉿kali)-[/home/kali/mininet/examples]
└─# python q3Script.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
(1000.00Mbit 1.00000% loss) (1000.00Mbit 1.00000% loss) (1
000.00Mbit 1.00000% loss) (1000.00Mbit 1.00000% loss) ***
Starting network
*** Configuring hosts
h1 h2
*** Starting controllers
*** Starting switches
(1000.00Mbit 1.00000% loss) (1000.00Mbit 1.00000% loss) **
* Post configure switches and hosts
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 → h2
h2 → h1
*** Results: 0% dropped (2/2 received)
mininet> xterm h1 h2
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=21451>
<Host h2: h2-eth0:10.0.0.2 pid=21453>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=21446>
<Controller c0: 127.0.0.1:6633 pid=21434>
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet>
```

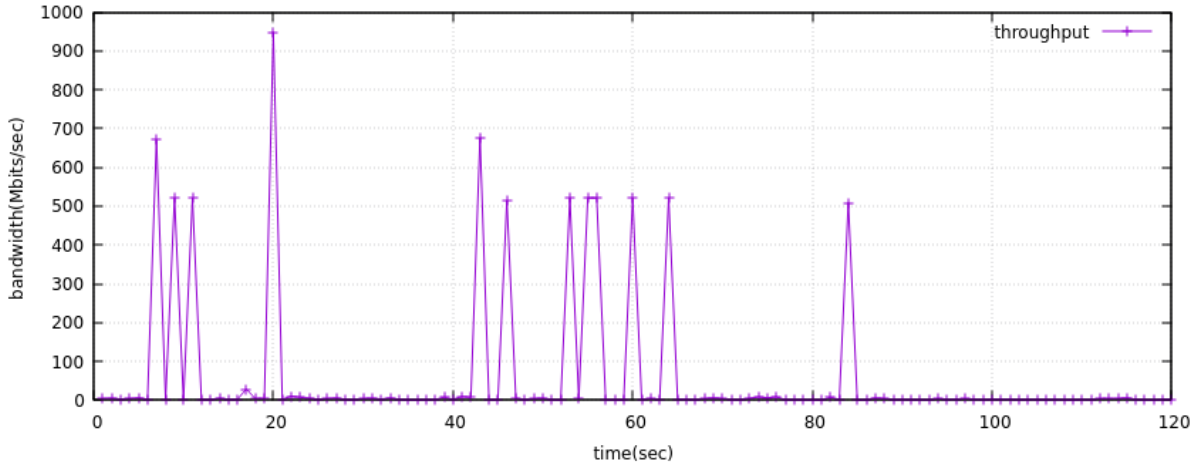
At 1% link loss:



At 2% link loss:



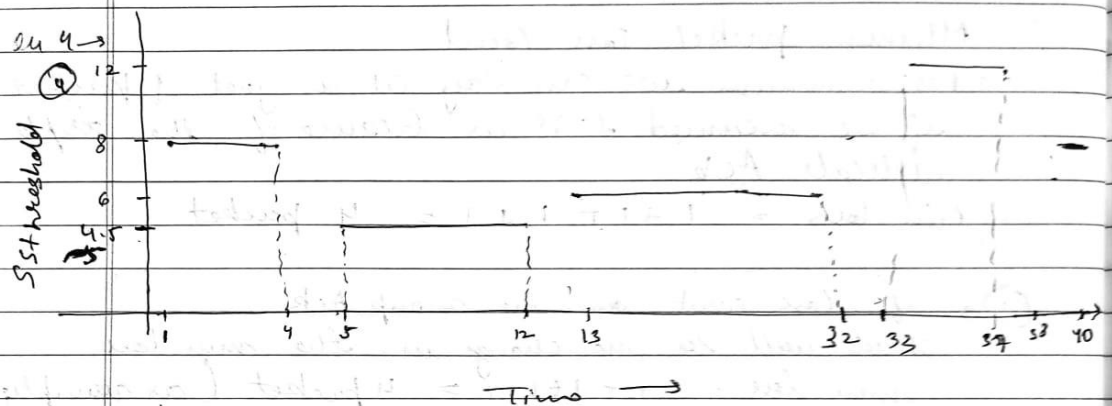
At 5% link loss:



	Bandwidth		
Loss (%)	h2(client)	h1(server)	Transfer (GB)
1	97.9	97.9	1.37
2	32.6	32.6	0.467
5	2.75	2.75	0.0395

- As the link loss increases the bandwidth decreases very quickly.
- The mininet reduces the size of file transfer also. It is because the loss increases the mininet tries to send the packet which will cause less dropage, this can be achieved by decreasing the size.

Question 4:



(a) 4.5 ~~≈~~ 5 as Window Size ∈ I

(b) If we TCP Tahoe was after cwnd is starting from 1 after loss event
at $t = 5, 13, 33, 38$ the cwnd = 1

(c) → From the graph we can see that in the interval 33 packet of 24 Size was sent, it causes the ~~packet~~ loss event but the source has sent the 24 Size packet & it was in the flight although loss occur

Page No.

apsara

Date:

Max packet = 24
Time interval = 33

(a) Maximum Packet loss count
→ if we say that the loss corresponds to time out then at the loss event the maximum number of packet lost is the congestion window size just before the loss happen.
9 + 12 + 24 + 16 = 61 packet

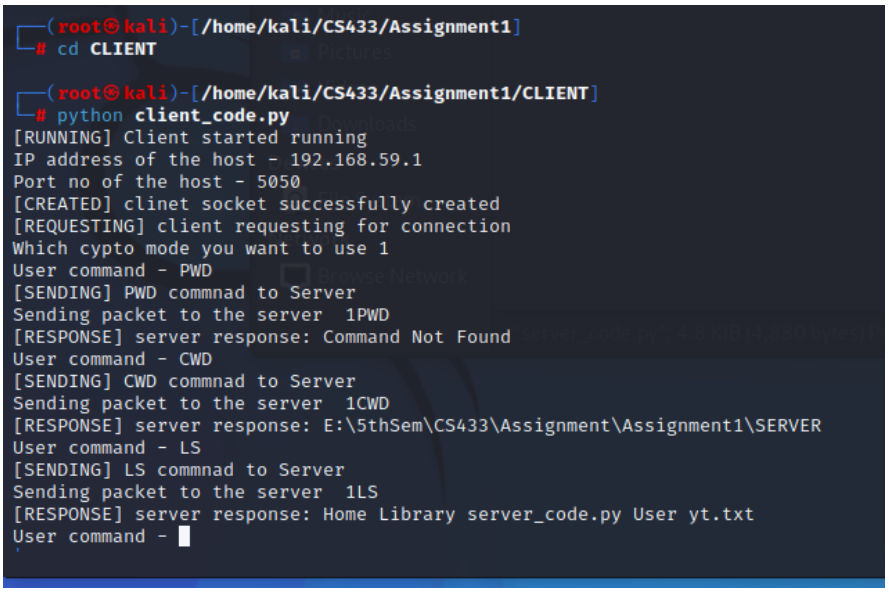
Minimum packet loss count
→ For minimum we can say it is just 1 packet as if we assumed loss is because of the triple duplicate Ack
Min loss = 1 + 1 + 1 + 1 = 4 packet

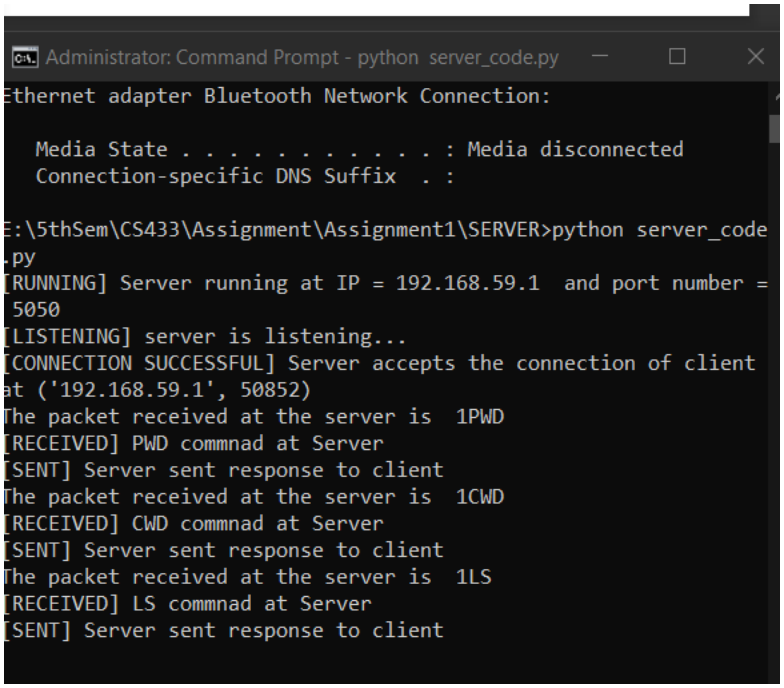
(b) → if loss event were due 3 dup Ack
there will be no change in the min loss
min loss = 1 + 1 + 1 + 1 = 4 packet (as assumption is valid
→ for max there will be a change as we have changed our assumption at 33 interval
Max loss = 9 + 12 + (24 - 3) + 16 = 58 packets
↳ due to 3 dup Ack.

Page No.:

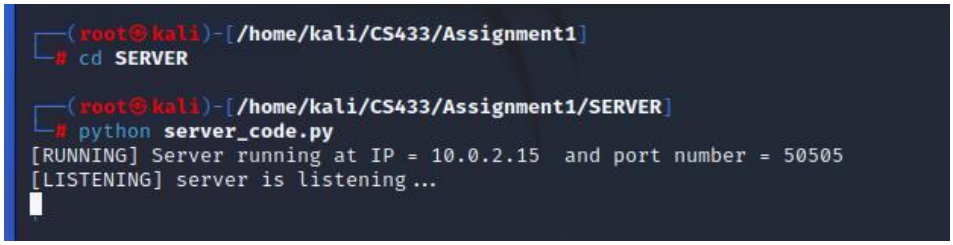
Grace Question:

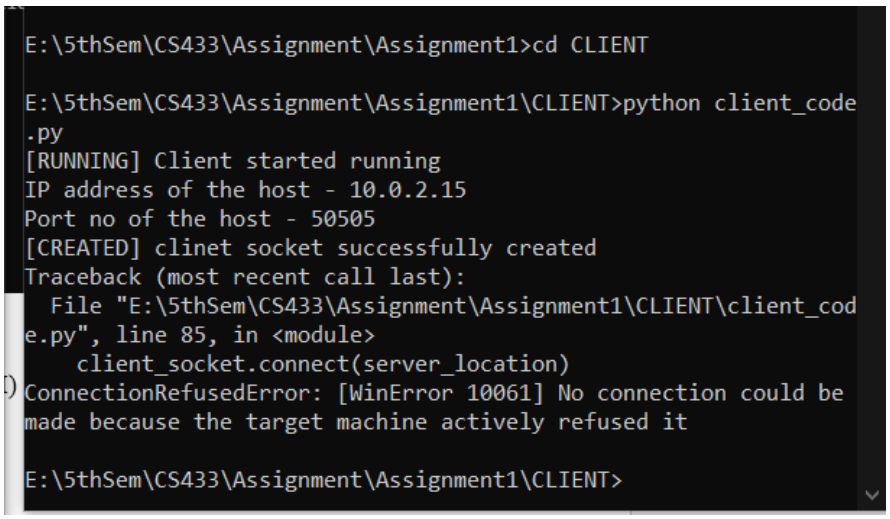
It worked when the server is on the host and client on the virtual machine





When I did reverse (client on the host and server on the VM) it didn't work, it says no connection can be made because the target machine actively refused it.





This is evident from the fact that NAT translation is one way. To make two we have to manually change the NAT translation.